Network Working Group
Request for Comments: 2251
Category: Standards Track

M. Wahl
Critical Angle Inc.
T. Howes
Netscape Communications Corp.
S. Kille
Isode Limited
December 1997

Lightweight Directory Access Protocol (v3)

1. Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

IESG Note

This document describes a directory access protocol that provides both read and update access. Update access requires secure authentication, but this document does not mandate implementation of any satisfactory authentication mechanisms.

In accordance with RFC 2026, section 4.4.1, this specification is being approved by IESG as a Proposed Standard despite this limitation, for the following reasons:

a. to encourage implementation and interoperability testing of these protocols (with or without update access) before they are deployed, and

b. to encourage deployment and use of these protocols in read-only applications. (e.g. applications where LDAPv3 is used as a query language for directories which are updated by some secure mechanism other than LDAP), and

c. to avoid delaying the advancement and deployment of other Internet standards-track protocols which require the ability to query, but not update, LDAPv3 directory servers.

Readers are hereby warned that until mandatory authentication mechanisms are standardized, clients and servers written according to this specification which make use of update functionality are UNLIKELY TO INTEROPERATE, or MAY INTEROPERATE ONLY IF AUTHENTICATION IS REDUCED TO AN UNACCEPTABLY WEAK LEVEL.

Implementors are hereby discouraged from deploying LDAPv3 clients or servers which implement the update functionality, until a Proposed Standard for mandatory authentication in LDAPv3 has been approved and published as an RFC.

Table of Contents

2. Abstract

The protocol described in this document is designed to provide access to directories supporting the X.500 models, while not incurring the resource requirements of the X.500 Directory Access Protocol (DAP). This protocol is specifically targeted at management applications and browser applications that provide read/write interactive access to directories. When used with a directory supporting the X.500 protocols, it is intended to be a complement to the X.500 DAP.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", and "MAY" in this document are to be interpreted as described in RFC 2119 [10].

Key aspects of this version of LDAP are:

- All protocol elements of LDAPv2 (RFC 1777) are supported. The protocol is carried directly over TCP or other transport, bypassing much of the session/presentation overhead of X.500 DAP.

- Most protocol data elements can be encoded as ordinary strings (e.g., Distinguished Names).

- Referrals to other servers may be returned.

- SASL mechanisms may be used with LDAP to provide association security services.

- Attribute values and Distinguished Names have been internationalized through the use of the ISO 10646 character set.

- The protocol can be extended to support new operations, and controls may be used to extend existing operations.

- Schema is published in the directory for use by clients.

3. Models

Interest in X.500 [1] directory technologies in the Internet has led to efforts to reduce the high cost of entry associated with use of these technologies. This document continues the efforts to define directory protocol alternatives, updating the LDAP [2] protocol specification.

3.1. Protocol Model

The general model adopted by this protocol is one of clients performing protocol operations against servers. In this model, a client transmits a protocol request describing the operation to be performed to a server. The server is then responsible for performing the necessary operation(s) in the directory. Upon completion of the operation(s), the server returns a response containing any results or errors to the requesting client.

In keeping with the goal of easing the costs associated with use of the directory, it is an objective of this protocol to minimize the complexity of clients so as to facilitate widespread deployment of applications capable of using the directory.

Note that although servers are required to return responses whenever such responses are defined in the protocol, there is no requirement for synchronous behavior on the part of either clients or servers. Requests and responses for multiple operations may be exchanged between a client and server in any order, provided the client eventually receives a response for every request that requires one.

In LDAP versions 1 and 2, no provision was made for protocol servers returning referrals to clients. However, for improved performance and distribution this version of the protocol permits servers to return to clients referrals to other servers. This allows servers to offload the work of contacting other servers to progress operations.

Note that the core protocol operations defined in this document can be mapped to a strict subset of the X.500(1997) directory abstract service, so it can be cleanly provided by the DAP. However there is not a one-to-one mapping between LDAP protocol operations and DAP operations: server implementations acting as a gateway to X.500 directories may need to make multiple DAP requests.

3.2. Data Model

This section provides a brief introduction to the X.500 data model, as used by LDAP.

The LDAP protocol assumes there are one or more servers which jointly provide access to a Directory Information Tree (DIT). The tree is made up of entries. Entries have names: one or more attribute values from the entry form its relative distinguished name (RDN), which MUST be unique among all its siblings. The concatenation of the relative distinguished names of the sequence of entries from a particular entry to an immediate subordinate of the root of the tree forms that entry's Distinguished Name (DN), which is unique in the tree. An example of a Distinguished Name is

CN=Steve Kille, O=Isode Limited, C=GB

Some servers may hold cache or shadow copies of entries, which can be used to answer search and comparison queries, but will return referrals or contact other servers if modification operations are requested.

Servers which perform caching or shadowing MUST ensure that they do not violate any access control constraints placed on the data by the originating server.

The largest collection of entries, starting at an entry that is mastered by a particular server, and including all its subordinates and their subordinates, down to the entries which are mastered by different servers, is termed a naming context. The root of the DIT is a DSA-specific Entry (DSE) and not part of any naming context: each server has different attribute values in the root DSE. (DSA is an X.500 term for the directory server).

3.2.1. Attributes of Entries

Entries consist of a set of attributes. An attribute is a type with one or more associated values. The attribute type is identified by a short descriptive name and an OID (object identifier). The attribute

type governs whether there can be more than one value of an attribute of that type in an entry, the syntax to which the values must conform, the kinds of matching which can be performed on values of that attribute, and other functions.

An example of an attribute is "mail". There may be one or more values of this attribute, they must be IA5 (ASCII) strings, and they are case insensitive (e.g. "foo@bar.com" will match "FOO@BAR.COM").

Schema is the collection of attribute type definitions, object class definitions and other information which a server uses to determine how to match a filter or attribute value assertion (in a compare operation) against the attributes of an entry, and whether to permit add and modify operations. The definition of schema for use with LDAP is given in [5] and [6]. Additional schema elements may be defined in other documents.

Each entry MUST have an objectClass attribute. The objectClass attribute specifies the object classes of an entry, which along with the system and user schema determine the permitted attributes of an entry. Values of this attribute may be modified by clients, but the objectClass attribute cannot be removed. Servers may restrict the modifications of this attribute to prevent the basic structural class of the entry from being changed (e.g. one cannot change a person into a country). When creating an entry or adding an objectClass value to an entry, all superclasses of the named classes are implicitly added as well if not already present, and the client must supply values for any mandatory attributes of new superclasses.

Some attributes, termed operational attributes, are used by servers for administering the directory system itself. They are not returned in search results unless explicitly requested by name. Attributes which are not operational, such as "mail", will have their schema and syntax constraints enforced by servers, but servers will generally not make use of their values.

Servers MUST NOT permit clients to add attributes to an entry unless those attributes are permitted by the object class definitions, the schema controlling that entry (specified in the subschema - see below), or are operational attributes known to that server and used for administrative purposes. Note that there is a particular objectClass 'extensibleObject' defined in [5] which permits all user attributes to be present in an entry.

Entries MAY contain, among others, the following operational attributes, defined in [5]. These attributes are maintained automatically by the server and are not modifiable by clients:

- creatorsName: the Distinguished Name of the user who added this entry to the directory.

- createTimestamp: the time this entry was added to the directory.

- modifiersName: the Distinguished Name of the user who last modified this entry.

- modifyTimestamp: the time this entry was last modified.

- subschemaSubentry: the Distinguished Name of the subschema entry (or subentry) which controls the schema for this entry.

3.2.2. Subschema Entries and Subentries

Subschema entries are used for administering information about the directory schema, in particular the object classes and attribute types supported by directory servers. A single subschema entry contains all schema definitions used by entries in a particular part of the directory tree.

Servers which follow X.500(93) models SHOULD implement subschema using the X.500 subschema mechanisms, and so these subschemas are not ordinary entries. LDAP clients SHOULD NOT assume that servers implement any of the other aspects of X.500 subschema. A server which masters entries and permits clients to modify these entries MUST implement and provide access to these subschema entries, so that its clients may discover the attributes and object classes which are permitted to be present. It is strongly recommended that all other servers implement this as well.

The following four attributes MUST be present in all subschema entries:

- cn: this attribute MUST be used to form the RDN of the subschema entry.

- objectClass: the attribute MUST have at least the values "top" and "subschema".

- objectClasses: each value of this attribute specifies an object class known to the server.

- attributeTypes: each value of this attribute specifies an attribute type known to the server.

These are defined in [5]. Other attributes MAY be present in subschema entries, to reflect additional supported capabilities.

These include matchingRules, matchingRuleUse, dITStructureRules, dITContentRules, nameForms and ldapSyntaxes.

Servers SHOULD provide the attributes createTimestamp and modifyTimestamp in subschema entries, in order to allow clients to maintain their caches of schema information.

Clients MUST only retrieve attributes from a subschema entry by requesting a base object search of the entry, where the search filter is "(objectClass=subschema)". (This will allow LDAPv3 servers which gateway to X.500(93) to detect that subentry information is being requested.)

3.3. Relationship to X.500

This document defines LDAP in terms of X.500 as an X.500 access mechanism. An LDAP server MUST act in accordance with the X.500(1993) series of ITU recommendations when providing the service. However, it is not required that an LDAP server make use of any X.500 protocols in providing this service, e.g. LDAP can be mapped onto any other directory system so long as the X.500 data and service model as used in LDAP is not violated in the LDAP interface.

3.4. Server-specific Data Requirements

An LDAP server MUST provide information about itself and other information that is specific to each server. This is represented as a group of attributes located in the root DSE (DSA-Specific Entry), which is named with the zero-length LDAPDN. These attributes are retrievable if a client performs a base object search of the root with filter "(objectClass=*)", however they are subject to access control restrictions. The root DSE MUST NOT be included if the client performs a subtree search starting from the root.

Servers may allow clients to modify these attributes.

The following attributes of the root DSE are defined in section 5 of [5]. Additional attributes may be defined in other documents.

- namingContexts: naming contexts held in the server. Naming contexts are defined in section 17 of X.501 [6].

- subschemaSubentry: subschema entries (or subentries) known by this server.

- altServer: alternative servers in case this one is later unavailable.

- supportedExtension: list of supported extended operations.

- supportedControl: list of supported controls.

- supportedSASLMechanisms: list of supported SASL security features.

- supportedLDAPVersion: LDAP versions implemented by the server.

If the server does not master entries and does not know the locations of schema information, the subschemaSubentry attribute is not present in the root DSE. If the server masters directory entries under one or more schema rules, there may be any number of values of the subschemaSubentry attribute in the root DSE.

4. Elements of Protocol

The LDAP protocol is described using Abstract Syntax Notation 1 (ASN.1) [3], and is typically transferred using a subset of ASN.1 Basic Encoding Rules [11]. In order to support future extensions to this protocol, clients and servers MUST ignore elements of SEQUENCE encodings whose tags they do not recognize.

Note that unlike X.500, each change to the LDAP protocol other than through the extension mechanisms will have a different version number. A client will indicate the version it supports as part of the bind request, described in section 4.2. If a client has not sent a bind, the server MUST assume that version 3 is supported in the client (since version 2 required that the client bind first).

Clients may determine the protocol version a server supports by reading the supportedLDAPVersion attribute from the root DSE. Servers which implement version 3 or later versions MUST provide this attribute. Servers which only implement version 2 may not provide this attribute.

4.1. Common Elements

This section describes the LDAPMessage envelope PDU (Protocol Data Unit) format, as well as data type definitions which are used in the protocol operations.

4.1.1. Message Envelope

For the purposes of protocol exchanges, all protocol operations are encapsulated in a common envelope, the LDAPMessage, which is defined as follows:

```
LDAPMessage ::= SEQUENCE {
```



```
        messageID       MessageID,
        protocolOp      CHOICE {
                bindRequest     BindRequest,
                bindResponse    BindResponse,
                unbindRequest   UnbindRequest,
                searchRequest   SearchRequest,
                searchResEntry  SearchResultEntry,
                searchResDone   SearchResultDone,
                searchResRef    SearchResultReference,
                modifyRequest   ModifyRequest,
                modifyResponse  ModifyResponse,
                addRequest      AddRequest,
                addResponse     AddResponse,
                delRequest      DelRequest,
                delResponse     DelResponse,
                modDNRequest    ModifyDNRequest,
                modDNResponse   ModifyDNResponse,
                compareRequest  CompareRequest,
                compareResponse CompareResponse,
                abandonRequest  AbandonRequest,
                extendedReq     ExtendedRequest,
                extendedResp    ExtendedResponse },
        controls        [0] Controls OPTIONAL }

MessageID ::= INTEGER (0 .. maxInt)

maxInt INTEGER ::= 2147483647 -- (2^^31 - 1) --
```

The function of the LDAPMessage is to provide an envelope containing common fields required in all protocol exchanges. At this time the only common fields are the message ID and the controls.

If the server receives a PDU from the client in which the LDAPMessage SEQUENCE tag cannot be recognized, the messageID cannot be parsed, the tag of the protocolOp is not recognized as a request, or the encoding structures or lengths of data fields are found to be incorrect, then the server MUST return the notice of disconnection described in section 4.4.1, with resultCode protocolError, and immediately close the connection. In other cases that the server cannot parse the request received by the client, the server MUST return an appropriate response to the request, with the resultCode set to protocolError.

If the client receives a PDU from the server which cannot be parsed, the client may discard the PDU, or may abruptly close the connection.

The ASN.1 type Controls is defined in section 4.1.12.

4.1.1.1. Message ID

All LDAPMessage envelopes encapsulating responses contain the messageID value of the corresponding request LDAPMessage.

The message ID of a request MUST have a value different from the values of any other requests outstanding in the LDAP session of which this message is a part.

A client MUST NOT send a second request with the same message ID as an earlier request on the same connection if the client has not received the final response from the earlier request. Otherwise the behavior is undefined. Typical clients increment a counter for each request.

A client MUST NOT reuse the message id of an abandonRequest or of the abandoned operation until it has received a response from the server for another request invoked subsequent to the abandonRequest, as the abandonRequest itself does not have a response.

4.1.2. String Types

The LDAPString is a notational convenience to indicate that, although strings of LDAPString type encode as OCTET STRING types, the ISO 10646 [13] character set (a superset of Unicode) is used, encoded following the UTF-8 algorithm [14]. Note that in the UTF-8 algorithm characters which are the same as ASCII (0x0000 through 0x007F) are represented as that same ASCII character in a single byte. The other byte values are used to form a variable-length encoding of an arbitrary character.

```
LDAPString ::= OCTET STRING
```

The LDAPOID is a notational convenience to indicate that the permitted value of this string is a (UTF-8 encoded) dotted-decimal representation of an OBJECT IDENTIFIER.

```
LDAPOID ::= OCTET STRING
```

For example,

```
1.3.6.1.4.1.1466.1.2.3
```

4.1.3. Distinguished Name and Relative Distinguished Name

An LDAPDN and a RelativeLDAPDN are respectively defined to be the representation of a Distinguished Name and a Relative Distinguished Name after encoding according to the specification in [4], such that


```
<distinguished-name> ::= <name>

<relative-distinguished-name> ::= <name-component>
```

where <name> and <name-component> are as defined in [4].

```
LDAPDN ::= LDAPString

RelativeLDAPDN ::= LDAPString
```

Only Attribute Types can be present in a relative distinguished name component; the options of Attribute Descriptions (next section) MUST NOT be used in specifying distinguished names.

4.1.4. Attribute Type

An AttributeType takes on as its value the textual string associated with that AttributeType in its specification.

```
AttributeType ::= LDAPString
```

Each attribute type has a unique OBJECT IDENTIFIER which has been assigned to it. This identifier may be written as decimal digits with components separated by periods, e.g. "2.5.4.10".

A specification may also assign one or more textual names for an attribute type. These names MUST begin with a letter, and only contain ASCII letters, digit characters and hyphens. They are case insensitive. (These ASCII characters are identical to ISO 10646 characters whose UTF-8 encoding is a single byte between 0x00 and 0x7F.)

If the server has a textual name for an attribute type, it MUST use a textual name for attributes returned in search results. The dotted-decimal OBJECT IDENTIFIER is only used if there is no textual name for an attribute type.

Attribute type textual names are non-unique, as two different specifications (neither in standards track RFCs) may choose the same name.

A server which masters or shadows entries SHOULD list all the attribute types it supports in the subschema entries, using the attributeTypes attribute. Servers which support an open-ended set of attributes SHOULD include at least the attributeTypes value for the 'objectClass' attribute. Clients MAY retrieve the attributeTypes value from subschema entries in order to obtain the OBJECT IDENTIFIER and other information associated with attribute types.

Some attribute type names which are used in this version of LDAP are described in [5]. Servers may implement additional attribute types.

4.1.5. Attribute Description

An AttributeDescription is a superset of the definition of the AttributeType. It has the same ASN.1 definition, but allows additional options to be specified. They are also case insensitive.

```
AttributeDescription ::= LDAPString
```

A value of AttributeDescription is based on the following BNF:

```
<AttributeDescription> ::= <AttributeType> [ ";" <options> ]

<options>  ::= <option> | <option> ";" <options>

<option>   ::= <opt-char> <opt-char>*

<opt-char> ::=  ASCII-equivalent letters, numbers and hyphen
```

Examples of valid AttributeDescription:

```
cn
userCertificate;binary
```

One option, "binary", is defined in this document. Additional options may be defined in IETF standards-track and experimental RFCs. Options beginning with "x-" are reserved for private experiments. Any option could be associated with any AttributeType, although not all combinations may be supported by a server.

An AttributeDescription with one or more options is treated as a subtype of the attribute type without any options. Options present in an AttributeDescription are never mutually exclusive. Implementations MUST generate the <options> list sorted in ascending order, and servers MUST treat any two AttributeDescription with the same AttributeType and options as equivalent. A server will treat an AttributeDescription with any options it does not implement as an unrecognized attribute type.

The data type "AttributeDescriptionList" describes a list of 0 or more attribute types. (A list of zero elements has special significance in the Search request.)

```
AttributeDescriptionList ::= SEQUENCE OF
        AttributeDescription
```


4.1.5.1. Binary Option

If the "binary" option is present in an AttributeDescription, it overrides any string-based encoding representation defined for that attribute in [5]. Instead the attribute is to be transferred as a binary value encoded using the Basic Encoding Rules [11]. The syntax of the binary value is an ASN.1 data type definition which is referenced by the "SYNTAX" part of the attribute type definition.

The presence or absence of the "binary" option only affects the transfer of attribute values in protocol; servers store any particular attribute in a single format. If a client requests that a server return an attribute in the binary format, but the server cannot generate that format, the server MUST treat this attribute type as an unrecognized attribute type. Similarly, clients MUST NOT expect servers to return an attribute in binary format if the client requested that attribute by name without the binary option.

This option is intended to be used with attributes whose syntax is a complex ASN.1 data type, and the structure of values of that type is needed by clients. Examples of this kind of syntax are "Certificate" and "CertificateList".

4.1.6. Attribute Value

A field of type AttributeValue takes on as its value either a string encoding of a AttributeValue data type, or an OCTET STRING containing an encoded binary value, depending on whether the "binary" option is present in the companion AttributeDescription to this AttributeValue.

The definition of string encodings for different syntaxes and types may be found in other documents, and in particular [5].

```
AttributeValue ::= OCTET STRING
```

Note that there is no defined limit on the size of this encoding; thus protocol values may include multi-megabyte attributes (e.g. photographs).

Attributes may be defined which have arbitrary and non-printable syntax. Implementations MUST NEITHER simply display nor attempt to decode as ASN.1 a value if its syntax is not known. The implementation may attempt to discover the subschema of the source entry, and retrieve the values of attributeTypes from it.

Clients MUST NOT send attribute values in a request which are not valid according to the syntax defined for the attributes.

4.1.7. Attribute Value Assertion

The AttributeValueAssertion type definition is similar to the one in the X.500 directory standards. It contains an attribute description and a matching rule assertion value suitable for that type.

```
AttributeValueAssertion ::= SEQUENCE {
        attributeDesc   AttributeDescription,
        assertionValue  AssertionValue }

AssertionValue ::= OCTET STRING
```

If the "binary" option is present in attributeDesc, this signals to the server that the assertionValue is a binary encoding of the assertion value.

For all the string-valued user attributes described in [5], the assertion value syntax is the same as the value syntax. Clients may use attribute values as assertion values in compare requests and search filters.

Note however that the assertion syntax may be different from the value syntax for other attributes or for non-equality matching rules. These may have an assertion syntax which contains only part of the value. See section 20.2.1.8 of X.501 [6] for examples.

4.1.8. Attribute

An attribute consists of a type and one or more values of that type. (Though attributes MUST have at least one value when stored, due to access control restrictions the set may be empty when transferred in protocol. This is described in section 4.5.2, concerning the PartialAttributeList type.)

```
Attribute ::= SEQUENCE {
        type    AttributeDescription,
        vals    SET OF AttributeValue }
```

Each attribute value is distinct in the set (no duplicates). The order of attribute values within the vals set is undefined and implementation-dependent, and MUST NOT be relied upon.

4.1.9. Matching Rule Identifier

A matching rule is a means of expressing how a server should compare an AssertionValue received in a search filter with an abstract data value. The matching rule defines the syntax of the assertion value and the process to be performed in the server.

An X.501(1993) Matching Rule is identified in the LDAP protocol by the printable representation of its OBJECT IDENTIFIER, either as one of the strings given in [5], or as decimal digits with components separated by periods, e.g. "caseIgnoreIA5Match" or "1.3.6.1.4.1.453.33.33".

```
MatchingRuleId ::= LDAPString
```

Servers which support matching rules for use in the extensibleMatch search filter MUST list the matching rules they implement in subschema entries, using the matchingRules attributes. The server SHOULD also list there, using the matchingRuleUse attribute, the attribute types with which each matching rule can be used. More information is given in section 4.4 of [5].

4.1.10. Result Message

The LDAPResult is the construct used in this protocol to return success or failure indications from servers to clients. In response to various requests servers will return responses containing fields of type LDAPResult to indicate the final status of a protocol operation request.

```
LDAPResult ::= SEQUENCE {
        resultCode      ENUMERATED {
                        success                      (0),
                        operationsError              (1),
                        protocolError                (2),
                        timeLimitExceeded            (3),
                        sizeLimitExceeded            (4),
                        compareFalse                 (5),
                        compareTrue                  (6),

                        authMethodNotSupported       (7),
                        strongAuthRequired           (8),
                                -- 9 reserved --
                        referral                     (10),  -- new
                        adminLimitExceeded           (11),  -- new
                        unavailableCriticalExtension (12),  -- new
                        confidentialityRequired      (13),  -- new
                        saslBindInProgress           (14),  -- new
                        noSuchAttribute              (16),
                        undefinedAttributeType       (17),
                        inappropriateMatching        (18),
                        constraintViolation          (19),
                        attributeOrValueExists       (20),
                        invalidAttributeSyntax       (21),
                                -- 22-31 unused --
```

```
                    noSuchObject                (32),
                    aliasProblem                (33),
                    invalidDNSyntax             (34),
                    -- 35 reserved for undefined isLeaf --
                    aliasDereferencingProblem   (36),
                              -- 37-47 unused --
                    inappropriateAuthentication (48),
                    invalidCredentials          (49),
                    insufficientAccessRights    (50),
                    busy                        (51),
                    unavailable                 (52),
                    unwillingToPerform          (53),
                    loopDetect                  (54),
                              -- 55-63 unused --
                    namingViolation             (64),
                    objectClassViolation        (65),
                    notAllowedOnNonLeaf         (66),
                    notAllowedOnRDN             (67),
                    entryAlreadyExists          (68),
                    objectClassModsProhibited   (69),
                              -- 70 reserved for CLDAP --
                    affectsMultipleDSAs         (71), -- new
                              -- 72-79 unused --
                    other                       (80) },
                    -- 81-90 reserved for APIs --
          matchedDN       LDAPDN,
          errorMessage    LDAPString,
          referral        [3] Referral OPTIONAL }
```

All the result codes with the exception of success, compareFalse and compareTrue are to be treated as meaning the operation could not be completed in its entirety.

Most of the result codes are based on problem indications from X.511 error data types. Result codes from 16 to 21 indicate an AttributeProblem, codes 32, 33, 34 and 36 indicate a NameProblem, codes 48, 49 and 50 indicate a SecurityProblem, codes 51 to 54 indicate a ServiceProblem, and codes 64 to 69 and 71 indicates an UpdateProblem.

If a client receives a result code which is not listed above, it is to be treated as an unknown error condition.

The errorMessage field of this construct may, at the server's option, be used to return a string containing a textual, human-readable (terminal control and page formatting characters should be avoided) error diagnostic. As this error diagnostic is not standardized,

implementations MUST NOT rely on the values returned. If the server chooses not to return a textual diagnostic, the errorMessage field of the LDAPResult type MUST contain a zero length string.

For result codes of noSuchObject, aliasProblem, invalidDNSyntax and aliasDereferencingProblem, the matchedDN field is set to the name of the lowest entry (object or alias) in the directory that was matched. If no aliases were dereferenced while attempting to locate the entry, this will be a truncated form of the name provided, or if aliases were dereferenced, of the resulting name, as defined in section 12.5 of X.511 [8]. The matchedDN field is to be set to a zero length string with all other result codes.

4.1.11. Referral

The referral error indicates that the contacted server does not hold the target entry of the request. The referral field is present in an LDAPResult if the LDAPResult.resultCode field value is referral, and absent with all other result codes. It contains a reference to another server (or set of servers) which may be accessed via LDAP or other protocols. Referrals can be returned in response to any operation request (except unbind and abandon which do not have responses). At least one URL MUST be present in the Referral.

The referral is not returned for a singleLevel or wholeSubtree search in which the search scope spans multiple naming contexts, and several different servers would need to be contacted to complete the operation. Instead, continuation references, described in section 4.5.3, are returned.

```
Referral ::= SEQUENCE OF LDAPURL  -- one or more

LDAPURL ::= LDAPString -- limited to characters permitted in URLs
```

If the client wishes to progress the operation, it MUST follow the referral by contacting any one of servers. All the URLs MUST be equally capable of being used to progress the operation. (The mechanisms for how this is achieved by multiple servers are outside the scope of this document.)

URLs for servers implementing the LDAP protocol are written according to [9]. If an alias was dereferenced, the <dn> part of the URL MUST be present, with the new target object name. If the <dn> part is present, the client MUST use this name in its next request to progress the operation, and if it is not present the client will use the same name as in the original request. Some servers (e.g. participating in distributed indexing) may provide a different filter in a referral for a search operation. If the filter part of the URL

is present in an LDAPURL, the client MUST use this filter in its next request to progress this search, and if it is not present the client MUST use the same filter as it used for that search. Other aspects of the new request may be the same or different as the request which generated the referral.

Note that UTF-8 characters appearing in a DN or search filter may not be legal for URLs (e.g. spaces) and MUST be escaped using the %

method in RFC 1738 [7].

Other kinds of URLs may be returned, so long as the operation could be performed using that protocol.

4.1.12. Controls

A control is a way to specify extension information. Controls which are sent as part of a request apply only to that request and are not saved.

```
Controls ::= SEQUENCE OF Control

Control ::= SEQUENCE {
        controlType             LDAPOID,
        criticality             BOOLEAN DEFAULT FALSE,
        controlValue            OCTET STRING OPTIONAL }
```

The controlType field MUST be a UTF-8 encoded dotted-decimal representation of an OBJECT IDENTIFIER which uniquely identifies the control. This prevents conflicts between control names.

The criticality field is either TRUE or FALSE.

If the server recognizes the control type and it is appropriate for the operation, the server will make use of the control when performing the operation.

If the server does not recognize the control type and the criticality field is TRUE, the server MUST NOT perform the operation, and MUST instead return the resultCode unsupportedCriticalExtension.

If the control is not appropriate for the operation and criticality field is TRUE, the server MUST NOT perform the operation, and MUST instead return the resultCode unsupportedCriticalExtension.

If the control is unrecognized or inappropriate but the criticality field is FALSE, the server MUST ignore the control.

The controlValue contains any information associated with the control, and its format is defined for the control. The server MUST be prepared to handle arbitrary contents of the controlValue octet string, including zero bytes. It is absent only if there is no value information which is associated with a control of its type.

This document does not define any controls. Controls may be defined in other documents. The definition of a control consists of:

- the OBJECT IDENTIFIER assigned to the control,

- whether the control is always noncritical, always critical, or

critical at the client's option,

- the format of the controlValue contents of the control.

Servers list the controls which they recognize in the supportedControl attribute in the root DSE.

4.2. Bind Operation

The function of the Bind Operation is to allow authentication information to be exchanged between the client and server.

The Bind Request is defined as follows:

```
BindRequest ::= [APPLICATION 0] SEQUENCE {
        version                 INTEGER (1 .. 127),
        name                    LDAPDN,
        authentication          AuthenticationChoice }

AuthenticationChoice ::= CHOICE {
        simple                  [0] OCTET STRING,
                                 -- 1 and 2 reserved
        sasl                    [3] SaslCredentials }

SaslCredentials ::= SEQUENCE {
        mechanism               LDAPString,
        credentials             OCTET STRING OPTIONAL }
```

Parameters of the Bind Request are:

- version: A version number indicating the version of the protocol to be used in this protocol session. This document describes version 3 of the LDAP protocol. Note that there is no version negotiation, and the client just sets this parameter to the version it desires. If the client requests protocol version 2, a server that supports the version 2 protocol as described in [2] will not return any v3-

specific protocol fields. (Note that not all LDAP servers will support protocol version 2, since they may be unable to generate the attribute syntaxes associated with version 2.)

- name: The name of the directory object that the client wishes to bind as. This field may take on a null value (a zero length string) for the purposes of anonymous binds, when authentication has been performed at a lower layer, or when using SASL credentials with a mechanism that includes the LDAPDN in the credentials.

- authentication: information used to authenticate the name, if any, provided in the Bind Request.

Upon receipt of a Bind Request, a protocol server will authenticate the requesting client, if necessary. The server will then return a Bind Response to the client indicating the status of the

authentication.

Authorization is the use of this authentication information when performing operations. Authorization MAY be affected by factors outside of the LDAP Bind request, such as lower layer security services.

4.2.1. Sequencing of the Bind Request

For some SASL authentication mechanisms, it may be necessary for the client to invoke the BindRequest multiple times. If at any stage the client wishes to abort the bind process it MAY unbind and then drop the underlying connection. Clients MUST NOT invoke operations between two Bind requests made as part of a multi-stage bind.

A client may abort a SASL bind negotiation by sending a BindRequest with a different value in the mechanism field of SaslCredentials, or an AuthenticationChoice other than sasl.

If the client sends a BindRequest with the sasl mechanism field as an empty string, the server MUST return a BindResponse with authMethodNotSupported as the resultCode. This will allow clients to abort a negotiation if it wishes to try again with the same SASL mechanism.

Unlike LDAP v2, the client need not send a Bind Request in the first PDU of the connection. The client may request any operations and the server MUST treat these as unauthenticated. If the server requires that the client bind before browsing or modifying the directory, the server MAY reject a request other than binding, unbinding or an extended request with the "operationsError" result.

If the client did not bind before sending a request and receives an operationsError, it may then send a Bind Request. If this also fails or the client chooses not to bind on the existing connection, it will close the connection, reopen it and begin again by first sending a PDU with a Bind Request. This will aid in interoperating with servers implementing other versions of LDAP.

Clients MAY send multiple bind requests on a connection to change their credentials. A subsequent bind process has the effect of abandoning all operations outstanding on the connection. (This simplifies server implementation.) Authentication from earlier binds are subsequently ignored, and so if the bind fails, the connection will be treated as anonymous. If a SASL transfer encryption or integrity mechanism has been negotiated, and that mechanism does not support the changing of credentials from one identity to another, then the client MUST instead establish a new connection.

4.2.2. Authentication and Other Security Services

The simple authentication option provides minimal authentication

facilities, with the contents of the authentication field consisting only of a cleartext password. Note that the use of cleartext passwords is not recommended over open networks when there is no authentication or encryption being performed by a lower layer; see the "Security Considerations" section.

If no authentication is to be performed, then the simple authentication option MUST be chosen, and the password be of zero length. (This is often done by LDAPv2 clients.) Typically the DN is also of zero length.

The sasl choice allows for any mechanism defined for use with SASL [12]. The mechanism field contains the name of the mechanism. The credentials field contains the arbitrary data used for authentication, inside an OCTET STRING wrapper. Note that unlike some Internet application protocols where SASL is used, LDAP is not text-based, thus no base64 transformations are performed on the credentials.

If any SASL-based integrity or confidentiality services are enabled, they take effect following the transmission by the server and reception by the client of the final BindResponse with resultCode success.

The client can request that the server use authentication information from a lower layer protocol by using the SASL EXTERNAL mechanism.

4.2.3. Bind Response

The Bind Response is defined as follows.

```
BindResponse ::= [APPLICATION 1] SEQUENCE {
     COMPONENTS OF LDAPResult,
     serverSaslCreds     [7] OCTET STRING OPTIONAL }
```

BindResponse consists simply of an indication from the server of he status of the client's request for authentication.

f the bind was successful, the resultCode will be success, therwise it will be one of:

- operationsError: server encountered an internal error,
- protocolError: unrecognized version number or incorrect PDU structure,
- authMethodNotSupported: unrecognized SASL mechanism name,
- strongAuthRequired: the server requires authentication be performed with a SASL mechanism,

- referral: this server cannot accept this bind and the client should try another,

- saslBindInProgress: the server requires the client to send a new bind request, with the same sasl mechanism, to continue the authentication process,

- inappropriateAuthentication: the server requires the client which had attempted to bind anonymously or without supplying credentials to provide some form of credentials,

- invalidCredentials: the wrong password was supplied or the SASL credentials could not be processed,

- unavailable: the server is shutting down.

If the server does not support the client's requested protocol version, it MUST set the resultCode to protocolError.

If the client receives a BindResponse response where the resultCode was protocolError, it MUST close the connection as the server will be unwilling to accept further operations. (This is for compatibility with earlier versions of LDAP, in which the bind was always the first operation, and there was no negotiation.)

The serverSaslCreds are used as part of a SASL-defined bind mechanism to allow the client to authenticate the server to which it is communicating, or to perform "challenge-response" authentication. If the client bound with the password choice, or the SASL mechanism does not require the server to return information to the client, then this field is not to be included in the result.

4.3. Unbind Operation

The function of the Unbind Operation is to terminate a protocol session. The Unbind Operation is defined as follows:

```
UnbindRequest ::= [APPLICATION 2] NULL
```

The Unbind Operation has no response defined. Upon transmission of an UnbindRequest, a protocol client may assume that the protocol session is terminated. Upon receipt of an UnbindRequest, a protocol server may assume that the requesting client has terminated the session and that all outstanding requests may be discarded, and may close the connection.

4.4. Unsolicited Notification

An unsolicited notification is an LDAPMessage sent from the server to the client which is not in response to any LDAPMessage received by the server. It is used to signal an extraordinary condition in the server or in the connection between the client and the server. The notification is of an advisory nature, and the server will not expect

any response to be returned from the client.

The unsolicited notification is structured as an LDAPMessage in which the messageID is 0 and protocolOp is of the extendedResp form. The responseName field of the ExtendedResponse is present. The LDAPOID value MUST be unique for this notification, and not be used in any other situation.

One unsolicited notification is defined in this document.

4.4.1. Notice of Disconnection

This notification may be used by the server to advise the client that the server is about to close the connection due to an error condition. Note that this notification is NOT a response to an unbind requested by the client: the server MUST follow the procedures of section 4.3. This notification is intended to assist clients in distinguishing between an error condition and a transient network

failure. As with a connection close due to network failure, the client MUST NOT assume that any outstanding requests which modified the directory have succeeded or failed.

The responseName is 1.3.6.1.4.1.1466.20036, the response field is absent, and the resultCode is used to indicate the reason for the disconnection.

The following resultCode values are to be used in this notification:

- protocolError: The server has received data from the client in which the LDAPMessage structure could not be parsed.

- strongAuthRequired: The server has detected that an established underlying security association protecting communication between the client and server has unexpectedly failed or been compromised.

- unavailable: This server will stop accepting new connections and operations on all existing connections, and be unavailable for an extended period of time. The client may make use of an alternative server.

After sending this notice, the server MUST close the connection. After receiving this notice, the client MUST NOT transmit any further on the connection, and may abruptly close the connection.

4.5. Search Operation

The Search Operation allows a client to request that a search be performed on its behalf by a server. This can be used to read attributes from a single entry, from entries immediately below a

particular entry, or a whole subtree of entries.

4.5.1. Search Request

The Search Request is defined as follows:

```
SearchRequest ::= [APPLICATION 3] SEQUENCE {
        baseObject      LDAPDN,
        scope           ENUMERATED {
                baseObject              (0),
                singleLevel             (1),
                wholeSubtree            (2) },
        derefAliases    ENUMERATED {
                neverDerefAliases       (0),
                derefInSearching        (1),
                derefFindingBaseObj     (2),
```



```
                derefAlways             (3) },
        sizeLimit       INTEGER (0 .. maxInt),
        timeLimit       INTEGER (0 .. maxInt),
        typesOnly       BOOLEAN,
        filter          Filter,
        attributes      AttributeDescriptionList }

Filter ::= CHOICE {
        and             [0] SET OF Filter,
        or              [1] SET OF Filter,
        not             [2] Filter,
        equalityMatch   [3] AttributeValueAssertion,
        substrings      [4] SubstringFilter,
        greaterOrEqual  [5] AttributeValueAssertion,
        lessOrEqual     [6] AttributeValueAssertion,
        present         [7] AttributeDescription,
        approxMatch     [8] AttributeValueAssertion,
        extensibleMatch [9] MatchingRuleAssertion }

SubstringFilter ::= SEQUENCE {
        type            AttributeDescription,
        -- at least one must be present
        substrings      SEQUENCE OF CHOICE {
                initial [0] LDAPString,
                any     [1] LDAPString,
                final   [2] LDAPString } }

MatchingRuleAssertion ::= SEQUENCE {
        matchingRule    [1] MatchingRuleId OPTIONAL,
        type            [2] AttributeDescription OPTIONAL,
        matchValue      [3] AssertionValue,
        dnAttributes    [4] BOOLEAN DEFAULT FALSE }
```

Parameters of the Search Request are:

- baseObject: An LDAPDN that is the base object entry relative to

which the search is to be performed.

- scope: An indicator of the scope of the search to be performed. The semantics of the possible values of this field are identical to the semantics of the scope field in the X.511 Search Operation.

- derefAliases: An indicator as to how alias objects (as defined in X.501) are to be handled in searching. The semantics of the possible values of this field are:

  neverDerefAliases: do not dereference aliases in searching or in locating the base object of the search;

  derefInSearching: dereference aliases in subordinates of the base object in searching, but not in locating the base object of the search;

  derefFindingBaseObj: dereference aliases in locating the base object of the search, but not when searching subordinates of the base object;

  derefAlways: dereference aliases both in searching and in locating the base object of the search.

- sizelimit: A sizelimit that restricts the maximum number of entries to be returned as a result of the search. A value of 0 in this field indicates that no client-requested sizelimit restrictions are in effect for the search. Servers may enforce a maximum number of entries to return.

- timelimit: A timelimit that restricts the maximum time (in seconds) allowed for a search. A value of 0 in this field indicates that no client-requested timelimit restrictions are in effect for the search.

- typesOnly: An indicator as to whether search results will contain both attribute types and values, or just attribute types. Setting this field to TRUE causes only attribute types (no values) to be returned. Setting this field to FALSE causes both attribute types and values to be returned.

- filter: A filter that defines the conditions that must be fulfilled in order for the search to match a given entry.

  The 'and', 'or' and 'not' choices can be used to form combinations of filters. At least one filter element MUST be present in an 'and' or 'or' choice. The others match against individual attribute values of entries in the scope of the search. (Implementor's note: the 'not' filter is an example of a tagged choice in an implicitly-tagged module. In BER this is treated as if the tag was explicit.)

  A server MUST evaluate filters according to the three-valued logic of X.511(93) section 7.8.1. In summary, a filter is evaluated to

either "TRUE", "FALSE" or "Undefined". If the filter evaluates to TRUE for a particular entry, then the attributes of that entry are returned as part of the search result (subject to any applicable access control restrictions). If the filter evaluates to FALSE or Undefined, then the entry is ignored for the search.

A filter of the "and" choice is TRUE if all the filters in the SET OF evaluate to TRUE, FALSE if at least one filter is FALSE, and otherwise Undefined. A filter of the "or" choice is FALSE if all of the filters in the SET OF evaluate to FALSE, TRUE if at least one filter is TRUE, and Undefined otherwise. A filter of the "not" choice is TRUE if the filter being negated is FALSE, FALSE if it is TRUE, and Undefined if it is Undefined.

The present match evaluates to TRUE where there is an attribute or subtype of the specified attribute description present in an entry, and FALSE otherwise (including a presence test with an unrecognized attribute description.)

The extensibleMatch is new in this version of LDAP. If the matchingRule field is absent, the type field MUST be present, and the equality match is performed for that type. If the type field is absent and matchingRule is present, the matchValue is compared against all attributes in an entry which support that matchingRule, and the matchingRule determines the syntax for the assertion value (the filter item evaluates to TRUE if it matches with at least one attribute in the entry, FALSE if it does not match any attribute in the entry, and Undefined if the matchingRule is not recognized or the assertionValue cannot be parsed.) If the type field is present and matchingRule is present, the matchingRule MUST be one permitted for use with that type, otherwise the filter item is undefined. If the dnAttributes field is set to TRUE, the match is applied against all the attributes in an entry's distinguished name as well, and also evaluates to TRUE if there is at least one attribute in the distinguished name for which the filter item evaluates to TRUE. (Editors note: The dnAttributes field is present so that there does not need to be multiple versions of generic matching rules such as for word matching, one to apply to entries and another to apply to entries and dn attributes as well).

A filter item evaluates to Undefined when the server would not be able to determine whether the assertion value matches an entry. If an attribute description in an equalityMatch, substrings, greaterOrEqual, lessOrEqual, approxMatch or extensibleMatch filter is not recognized by the server, a matching rule id in the extensibleMatch is not recognized by the server, the assertion value cannot be parsed, or the type of filtering requested is not implemented, then the filter is Undefined. Thus for example if a server did not recognize the attribute type shoeSize, a filter of (shoeSize=*) would evaluate to FALSE, and the filters (shoeSize=12),

(shoeSize>=12) and (shoeSize<=12) would evaluate to Undefined.

Servers MUST NOT return errors if attribute descriptions or matching rule ids are not recognized, or assertion values cannot be parsed. More details of filter processing are given in section 7.8 of X.511 [8].

- attributes: A list of the attributes to be returned from each entry which matches the search filter. There are two special values which may be used: an empty list with no attributes, and the attribute description string "*". Both of these signify that all user attributes are to be returned. (The "*" allows the client to request all user attributes in addition to specific operational attributes).

  Attributes MUST be named at most once in the list, and are returned at most once in an entry. If there are attribute descriptions in the list which are not recognized, they are ignored by the server.

  If the client does not want any attributes returned, it can specify a list containing only the attribute with OID "1.1". This OID was chosen arbitrarily and does not correspond to any attribute in use.

  Client implementors should note that even if all user attributes are requested, some attributes of the entry may not be included in search results due to access control or other restrictions. Furthermore, servers will not return operational attributes, such as objectClasses or attributeTypes, unless they are listed by name, since there may be extremely large number of values for certain operational attributes. (A list of operational attributes for use in LDAP is given in [5].)

Note that an X.500 "list"-like operation can be emulated by the client requesting a one-level LDAP search operation with a filter checking for the existence of the objectClass attribute, and that an X.500 "read"-like operation can be emulated by a base object LDAP search operation with the same filter. A server which provides a gateway to X.500 is not required to use the Read or List operations, although it may choose to do so, and if it does must provide the same semantics as the X.500 search operation.

4.5.2. Search Result

The results of the search attempted by the server upon receipt of a Search Request are returned in Search Responses, which are LDAP messages containing either SearchResultEntry, SearchResultReference, ExtendedResponse or SearchResultDone data types.

```
SearchResultEntry ::= [APPLICATION 4] SEQUENCE {
        objectName      LDAPDN,
```

```
        attributes      PartialAttributeList }

    PartialAttributeList ::= SEQUENCE OF SEQUENCE {
            type    AttributeDescription,
            vals    SET OF AttributeValue }
    -- implementors should note that the PartialAttributeList may
    -- have zero elements (if none of the attributes of that entry
    -- were requested, or could be returned), and that the vals set
    -- may also have zero elements (if types only was requested, or
    -- all values were excluded from the result.)

    SearchResultReference ::= [APPLICATION 19] SEQUENCE OF LDAPURL
    -- at least one LDAPURL element must be present

    SearchResultDone ::= [APPLICATION 5] LDAPResult
```

Upon receipt of a Search Request, a server will perform the necessary search of the DIT.

If the LDAP session is operating over a connection-oriented transport such as TCP, the server will return to the client a sequence of responses in separate LDAP messages. There may be zero or more responses containing SearchResultEntry, one for each entry found during the search. There may also be zero or more responses containing SearchResultReference, one for each area not explored by this server during the search. The SearchResultEntry and SearchResultReference PDUs may come in any order. Following all the SearchResultReference responses and all SearchResultEntry responses to be returned by the server, the server will return a response containing the SearchResultDone, which contains an indication of success, or detailing any errors that have occurred.

Each entry returned in a SearchResultEntry will contain all attributes, complete with associated values if necessary, as specified in the attributes field of the Search Request. Return of attributes is subject to access control and other administrative policy. Some attributes may be returned in binary format (indicated by the AttributeDescription in the response having the binary option present).

Some attributes may be constructed by the server and appear in a SearchResultEntry attribute list, although they are not stored attributes of an entry. Clients MUST NOT assume that all attributes can be modified, even if permitted by access control.

LDAPMessage responses of the ExtendedResponse form are reserved for returning information associated with a control requested by the client. These may be defined in future versions of this document.

4.5.3. Continuation References in the Search Result

If the server was able to locate the entry referred to by the baseObject but was unable to search all the entries in the scope at and under the baseObject, the server may return one or more SearchResultReference, each containing a reference to another set of servers for continuing the operation. A server MUST NOT return any SearchResultReference if it has not located the baseObject and thus has not searched any entries; in this case it would return a SearchResultDone containing a referral resultCode.

In the absence of indexing information provided to a server from servers holding subordinate naming contexts, SearchResultReference responses are not affected by search filters and are always returned when in scope.

The SearchResultReference is of the same data type as the Referral. URLs for servers implementing the LDAP protocol are written according to [9]. The <dn> part MUST be present in the URL, with the new target object name. The client MUST use this name in its next request. Some servers (e.g. part of a distributed index exchange system) may provide a different filter in the URLs of the SearchResultReference. If the filter part of the URL is present in an LDAP URL, the client MUST use the new filter in its next request to progress the search, and if the filter part is absent the client will use again the same filter. Other aspects of the new search request may be the same or different as the search which generated the continuation references.

Other kinds of URLs may be returned so long as the operation could be performed using that protocol.

The name of an unexplored subtree in a SearchResultReference need not be subordinate to the base object.

In order to complete the search, the client MUST issue a new search operation for each SearchResultReference that is returned. Note that the abandon operation described in section 4.11 applies only to a particular operation sent on a connection between a client and server, and if the client has multiple outstanding search operations to different servers, it MUST abandon each operation individually.

4.5.3.1. Example

For example, suppose the contacted server (hosta) holds the entry "O=MNN,C=WW" and the entry "CN=Manager,O=MNN,C=WW". It knows that either LDAP-capable servers (hostb) or (hostc) hold "OU=People,O=MNN,C=WW" (one is the master and the other server a

shadow), and that LDAP-capable server (hostd) holds the subtree "OU=Roles,O=MNN,C=WW". If a subtree search of "O=MNN,C=WW" is requested to the contacted server, it may return the following:

```
SearchResultEntry for O=MNN,C=WW
SearchResultEntry for CN=Manager,O=MNN,C=WW
SearchResultReference {
  ldap://hostb/OU=People,O=MNN,C=WW
  ldap://hostc/OU=People,O=MNN,C=WW
}
SearchResultReference {
  ldap://hostd/OU=Roles,O=MNN,C=WW
}
SearchResultDone (success)
```

Client implementors should note that when following a SearchResultReference, additional SearchResultReference may be generated. Continuing the example, if the client contacted the server (hostb) and issued the search for the subtree "OU=People,O=MNN,C=WW", the server might respond as follows:

```
SearchResultEntry for OU=People,O=MNN,C=WW
SearchResultReference {
 ldap://hoste/OU=Managers,OU=People,O=MNN,C=WW
}
SearchResultReference {
 ldap://hostf/OU=Consultants,OU=People,O=MNN,C=WW
}
SearchResultDone (success)
```

If the contacted server does not hold the base object for the search, then it will return a referral to the client. For example, if the client requests a subtree search of "O=XYZ,C=US" to hosta, the server may return only a SearchResultDone containing a referral.

```
SearchResultDone (referral) {
  ldap://hostg/
}
```

4.6. Modify Operation

The Modify Operation allows a client to request that a modification of an entry be performed on its behalf by a server. The Modify Request is defined as follows:

```
ModifyRequest ::= [APPLICATION 6] SEQUENCE {
        object          LDAPDN,
        modification    SEQUENCE OF SEQUENCE {

                operation       ENUMERATED {
                                        add     (0),
                                        delete  (1),
                                        replace (2) },
```

```
                    modification    AttributeTypeAndValues } }

    AttributeTypeAndValues ::= SEQUENCE {
            type    AttributeDescription,
            vals    SET OF AttributeValue }
```

Parameters of the Modify Request are:

- object: The object to be modified. The value of this field contains the DN of the entry to be modified. The server will not perform any alias dereferencing in determining the object to be modified.

- modification: A list of modifications to be performed on the entry. The entire list of entry modifications MUST be performed in the order they are listed, as a single atomic operation. While individual modifications may violate the directory schema, the resulting entry after the entire list of modifications is performed MUST conform to the requirements of the directory schema. The values that may be taken on by the 'operation' field in each modification construct have the following semantics respectively:

    add: add values listed to the given attribute, creating the attribute if necessary;

    delete: delete values listed from the given attribute, removing the entire attribute if no values are listed, or if all current values of the attribute are listed for deletion;

    replace: replace all existing values of the given attribute with the new values listed, creating the attribute if it did not already exist. A replace with no value will delete the entire attribute if it exists, and is ignored if the attribute does not exist.

The result of the modify attempted by the server upon receipt of a Modify Request is returned in a Modify Response, defined as follows:

```
    ModifyResponse ::= [APPLICATION 7] LDAPResult
```

Upon receipt of a Modify Request, a server will perform the necessary modifications to the DIT.

The server will return to the client a single Modify Response indicating either the successful completion of the DIT modification, or the reason that the modification failed. Note that due to the requirement for atomicity in applying the list of modifications in the Modify Request, the client may expect that no modifications of the DIT have been performed if the Modify Response received indicates any sort of error, and that all requested modifications have been performed if the Modify Response indicates successful completion of

the Modify Operation. If the connection fails, whether the modification occurred or not is indeterminate.

The Modify Operation cannot be used to remove from an entry any of its distinguished values, those values which form the entry's relative distinguished name. An attempt to do so will result in the server returning the error notAllowedOnRDN. The Modify DN Operation described in section 4.9 is used to rename an entry.

If an equality match filter has not been defined for an attribute type, clients MUST NOT attempt to delete individual values of that attribute from an entry using the "delete" form of a modification, and MUST instead use the "replace" form.

Note that due to the simplifications made in LDAP, there is not a direct mapping of the modifications in an LDAP ModifyRequest onto the EntryModifications of a DAP ModifyEntry operation, and different implementations of LDAP-DAP gateways may use different means of representing the change. If successful, the final effect of the operations on the entry MUST be identical.

4.7. Add Operation

The Add Operation allows a client to request the addition of an entry into the directory. The Add Request is defined as follows:

```
AddRequest ::= [APPLICATION 8] SEQUENCE {
        entry           LDAPDN,
        attributes      AttributeList }

AttributeList ::= SEQUENCE OF SEQUENCE {
        type    AttributeDescription,
        vals    SET OF AttributeValue }
```

Parameters of the Add Request are:

- entry: the Distinguished Name of the entry to be added. Note that the server will not dereference any aliases in locating the entry to be added.

- attributes: the list of attributes that make up the content of the entry being added. Clients MUST include distinguished values (those forming the entry's own RDN) in this list, the objectClass attribute, and values of any mandatory attributes of the listed object classes. Clients MUST NOT supply the createTimestamp or creatorsName attributes, since these will be generated automatically by the server.

The entry named in the entry field of the AddRequest MUST NOT exist for the AddRequest to succeed. The parent of the entry to be added MUST exist. For example, if the client attempted to add "CN=JS,O=Foo,C=US", the "O=Foo,C=US" entry did not exist, and the

"C=US" entry did exist, then the server would return the error noSuchObject with the matchedDN field containing "C=US". If the parent entry exists but is not in a naming context held by the server, the server SHOULD return a referral to the server holding the parent entry.

Servers implementations SHOULD NOT restrict where entries can be located in the directory. Some servers MAY allow the administrator to restrict the classes of entries which can be added to the directory.

Upon receipt of an Add Request, a server will attempt to perform the add requested. The result of the add attempt will be returned to the client in the Add Response, defined as follows:

```
AddResponse ::= [APPLICATION 9] LDAPResult
```

A response of success indicates that the new entry is present in the directory.

4.8. Delete Operation

The Delete Operation allows a client to request the removal of an entry from the directory. The Delete Request is defined as follows:

```
DelRequest ::= [APPLICATION 10] LDAPDN
```

The Delete Request consists of the Distinguished Name of the entry to be deleted. Note that the server will not dereference aliases while resolving the name of the target entry to be removed, and that only leaf entries (those with no subordinate entries) can be deleted with this operation.

The result of the delete attempted by the server upon receipt of a Delete Request is returned in the Delete Response, defined as follows:


```
DelResponse ::= [APPLICATION 11] LDAPResult
```

Upon receipt of a Delete Request, a server will attempt to perform the entry removal requested. The result of the delete attempt will be returned to the client in the Delete Response.

4.9. Modify DN Operation

The Modify DN Operation allows a client to change the leftmost (least significant) component of the name of an entry in the directory, or to move a subtree of entries to a new location in the directory. The Modify DN Request is defined as follows:

```
ModifyDNRequest ::= [APPLICATION 12] SEQUENCE {
        entry           LDAPDN,
        newrdn          RelativeLDAPDN,
```

```
        deleteoldrdn    BOOLEAN,
        newSuperior     [0] LDAPDN OPTIONAL }
```

Parameters of the Modify DN Request are:

- entry: the Distinguished Name of the entry to be changed. This entry may or may not have subordinate entries.

- newrdn: the RDN that will form the leftmost component of the new name of the entry.

- deleteoldrdn: a boolean parameter that controls whether the old RDN attribute values are to be retained as attributes of the entry, or deleted from the entry.

- newSuperior: if present, this is the Distinguished Name of the entry which becomes the immediate superior of the existing entry.

The result of the name change attempted by the server upon receipt of a Modify DN Request is returned in the Modify DN Response, defined as follows:

```
    ModifyDNResponse ::= [APPLICATION 13] LDAPResult
```

Upon receipt of a ModifyDNRequest, a server will attempt to perform the name change. The result of the name change attempt will be returned to the client in the Modify DN Response.

For example, if the entry named in the "entry" parameter was "cn=John Smith,c=US", the newrdn parameter was "cn=John Cougar Smith", and the newSuperior parameter was absent, then this operation would

attempt to rename the entry to be "cn=John Cougar Smith,c=US". If there was already an entry with that name, the operation would fail with error code entryAlreadyExists.

If the deleteoldrdn parameter is TRUE, the values forming the old RDN are deleted from the entry. If the deleteoldrdn parameter is FALSE, the values forming the old RDN will be retained as non-distinguished attribute values of the entry. The server may not perform the operation and return an error code if the setting of the deleteoldrdn parameter would cause a schema inconsistency in the entry.

Note that X.500 restricts the ModifyDN operation to only affect entries that are contained within a single server. If the LDAP server is mapped onto DAP, then this restriction will apply, and the resultCode affectsMultipleDSAs will be returned if this error occurred. In general clients MUST NOT expect to be able to perform arbitrary movements of entries and subtrees between servers.

4.10. Compare Operation

The Compare Operation allows a client to compare an assertion provided with an entry in the directory. The Compare Request is defined as follows:

```
CompareRequest ::= [APPLICATION 14] SEQUENCE {
        entry           LDAPDN,
        ava             AttributeValueAssertion }
```

Parameters of the Compare Request are:

- entry: the name of the entry to be compared with.

- ava: the assertion with which an attribute in the entry is to be compared.

The result of the compare attempted by the server upon receipt of a Compare Request is returned in the Compare Response, defined as follows:

```
CompareResponse ::= [APPLICATION 15] LDAPResult
```

Upon receipt of a Compare Request, a server will attempt to perform the requested comparison. The result of the comparison will be returned to the client in the Compare Response. Note that errors and the result of comparison are all returned in the same construct.

Note that some directory systems may establish access controls which permit the values of certain attributes (such as userPassword) to be compared but not read. In a search result, it may be that an attribute of that type would be returned, but with an empty set of values.

4.11. Abandon Operation

The function of the Abandon Operation is to allow a client to request that the server abandon an outstanding operation. The Abandon Request is defined as follows:

```
AbandonRequest ::= [APPLICATION 16] MessageID
```

The MessageID MUST be that of a an operation which was requested earlier in this connection.

(The abandon request itself has its own message id. This is distinct from the id of the earlier operation being abandoned.)

There is no response defined in the Abandon Operation. Upon transmission of an Abandon Operation, a client may expect that the operation identified by the Message ID in the Abandon Request has been abandoned. In the event that a server receives an Abandon

Request on a Search Operation in the midst of transmitting responses to the search, that server MUST cease transmitting entry responses to the abandoned request immediately, and MUST NOT send the SearchResponseDone. Of course, the server MUST ensure that only properly encoded LDAPMessage PDUs are transmitted.

Clients MUST NOT send abandon requests for the same operation multiple times, and MUST also be prepared to receive results from operations it has abandoned (since these may have been in transit when the abandon was requested).

Servers MUST discard abandon requests for message IDs they do not recognize, for operations which cannot be abandoned, and for operations which have already been abandoned.

4.12. Extended Operation

An extension mechanism has been added in this version of LDAP, in order to allow additional operations to be defined for services not available elsewhere in this protocol, for instance digitally signed operations and results.

The extended operation allows clients to make requests and receive responses with predefined syntaxes and semantics. These may be defined in RFCs or be private to particular implementations. Each request MUST have a unique OBJECT IDENTIFIER assigned to it.

```
ExtendedRequest ::= [APPLICATION 23] SEQUENCE {
        requestName      [0] LDAPOID,
        requestValue     [1] OCTET STRING OPTIONAL }
```

The requestName is a dotted-decimal representation of the OBJECT IDENTIFIER corresponding to the request. The requestValue is information in a form defined by that request, encapsulated inside an OCTET STRING.

The server will respond to this with an LDAPMessage containing the ExtendedResponse.

```
ExtendedResponse ::= [APPLICATION 24] SEQUENCE {
        COMPONENTS OF LDAPResult,
        responseName     [10] LDAPOID OPTIONAL,
        response         [11] OCTET STRING OPTIONAL }
```

If the server does not recognize the request name, it MUST return only the response fields from LDAPResult, containing the protocolError result code.

5. Protocol Element Encodings and Transfer

One underlying service is defined here. Clients and servers SHOULD implement the mapping of LDAP over TCP described in 5.2.1.

5.1. Mapping Onto BER-based Transport Services

The protocol elements of LDAP are encoded for exchange using the Basic Encoding Rules (BER) [11] of ASN.1 [3]. However, due to the high overhead involved in using certain elements of the BER, the following additional restrictions are placed on BER-encodings of LDAP protocol elements:

(1) Only the definite form of length encoding will be used.

(2) OCTET STRING values will be encoded in the primitive form only.

(3) If the value of a BOOLEAN type is true, the encoding MUST have its contents octets set to hex "FF".

(4) If a value of a type is its default value, it MUST be absent. Only some BOOLEAN and INTEGER types have default values in this protocol definition.

These restrictions do not apply to ASN.1 types encapsulated inside of OCTET STRING values, such as attribute values, unless otherwise noted.

5.2. Transfer Protocols

This protocol is designed to run over connection-oriented, reliable transports, with all 8 bits in an octet being significant in the data stream.

5.2.1. Transmission Control Protocol (TCP)

The LDAPMessage PDUs are mapped directly onto the TCP bytestream. It is recommended that server implementations running over the TCP MAY provide a protocol listener on the assigned port, 389. Servers may instead provide a listener on a different port number. Clients MUST support contacting servers on any valid TCP port.

6. Implementation Guidelines

This document describes an Internet protocol.

6.1. Server Implementations

The server MUST be capable of recognizing all the mandatory attribute type names and implement the syntaxes specified in [5]. Servers MAY also recognize additional attribute type names.

6.2. Client Implementations

Clients which request referrals MUST ensure that they do not loop between servers. They MUST NOT repeatedly contact the same server for the same request with the same target entry name, scope and filter. Some clients may be using a counter that is incremented each time referral handling occurs for an operation, and these kinds of clients MUST be able to handle a DIT with at least ten layers of naming contexts between the root and a leaf entry.

In the absence of prior agreements with servers, clients SHOULD NOT assume that servers support any particular schemas beyond those referenced in section 6.1. Different schemas can have different attribute types with the same names. The client can retrieve the subschema entries referenced by the subschemaSubentry attribute in the server's root DSE or in entries held by the server.

7. Security Considerations

When used with a connection-oriented transport, this version of the protocol provides facilities for the LDAP v2 authentication mechanism, simple authentication using a cleartext password, as well as any SASL mechanism [12]. SASL allows for integrity and privacy services to be negotiated.

It is also permitted that the server can return its credentials to the client, if it chooses to do so.

Use of cleartext password is strongly discouraged where the underlying transport service cannot guarantee confidentiality and may result in disclosure of the password to unauthorized parties.

When used with SASL, it should be noted that the name field of the BindRequest is not protected against modification. Thus if the distinguished name of the client (an LDAPDN) is agreed through the negotiation of the credentials, it takes precedence over any value in the unprotected name field.

Implementations which cache attributes and entries obtained via LDAP MUST ensure that access controls are maintained if that information is to be provided to multiple clients, since servers may have access control policies which prevent the return of entries or attributes in search results except to particular authenticated clients. For example, caches could serve result information only to the client whose request caused it to be cache.

8. Acknowledgements

This document is an update to RFC 1777, by Wengyik Yeong, Tim Howes, and Steve Kille. Design ideas included in this document are based on those discussed in ASID and other IETF Working Groups. The contributions of individuals in these working groups is gratefully acknowledged.

9. Bibliography

[1] ITU-T Rec. X.500, "The Directory: Overview of Concepts, Models and Service", 1993.

[2] Yeong, W., Howes, T., and S. Kille, "Lightweight Directory Access Protocol", RFC 1777, March 1995.

[3] ITU-T Rec. X.680, "Abstract Syntax Notation One (ASN.1) - Specification of Basic Notation", 1994.

[4] Kille, S., Wahl, M., and T. Howes, "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC 2253, December 1997.

[5] Wahl, M., Coulbeck, A., Howes, T., and S. Kille, "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, December 1997.

[6] ITU-T Rec. X.501, "The Directory: Models", 1993.

[7] Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, December 1994.

[8] ITU-T Rec. X.511, "The Directory: Abstract Service Definition", 1993.

[9] Howes, T., and M. Smith, "The LDAP URL Format", RFC 2255, December 1997.

[10] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[11] ITU-T Rec. X.690, "Specification of ASN.1 encoding rules: Basic, Canonical, and Distinguished Encoding Rules", 1994.

[12] Meyers, J., "Simple Authentication and Security Layer", RFC 2222, October 1997.

[13] Universal Multiple-Octet Coded Character Set (UCS) - Architecture and Basic Multilingual Plane, ISO/IEC 10646-1 : 1993.

[14] Yergeau, F., "UTF-8, a transformation format of Unicode and ISO 10646", RFC 2044, October 1996.

10. Authors' Addresses

Mark Wahl
Critical Angle Inc.
4815 W Braker Lane #502-385

Austin, TX 78759
USA

Phone: +1 512 372-3160
EMail: M.Wahl@critical-angle.com

Tim Howes
Netscape Communications Corp.
501 E. Middlefield Rd., MS MV068
Mountain View, CA 94043
USA

Phone: +1 650 937-3419
EMail: howes@netscape.com

Steve Kille
Isode Limited
The Dome, The Square
Richmond
TW9 1DT
UK

Phone: +44-181-332-9091
EMail: S.Kille@isode.com

Appendix A - Complete ASN.1 Definition

```
        Lightweight-Directory-Access-Protocol-V3 DEFINITIONS
        IMPLICIT TAGS ::=

        BEGIN

        LDAPMessage ::= SEQUENCE {
                messageID       MessageID,
                protocolOp      CHOICE {
                        bindRequest     BindRequest,
                        bindResponse    BindResponse,
                        unbindRequest   UnbindRequest,
                        searchRequest   SearchRequest,
                        searchResEntry  SearchResultEntry,
                        searchResDone   SearchResultDone,
                        searchResRef    SearchResultReference,
                        modifyRequest   ModifyRequest,
                        modifyResponse  ModifyResponse,
                        addRequest      AddRequest,
                        addResponse     AddResponse,
                        delRequest      DelRequest,
                        delResponse     DelResponse,
                        modDNRequest    ModifyDNRequest,
                        modDNResponse   ModifyDNResponse,
                        compareRequest  CompareRequest,
                        compareResponse CompareResponse,
                        abandonRequest  AbandonRequest,
                        extendedReq     ExtendedRequest,
                        extendedResp    ExtendedResponse },
                 controls       [0] Controls OPTIONAL }

        MessageID ::= INTEGER (0 .. maxInt)

        maxInt INTEGER ::= 2147483647 -- (2^^31 - 1) --

        LDAPString ::= OCTET STRING

        LDAPOID ::= OCTET STRING

        LDAPDN ::= LDAPString

        RelativeLDAPDN ::= LDAPString

        AttributeType ::= LDAPString

        AttributeDescription ::= LDAPString
```

```
        AttributeDescriptionList ::= SEQUENCE OF
                AttributeDescription

        AttributeValue ::= OCTET STRING

        AttributeValueAssertion ::= SEQUENCE {
                attributeDesc   AttributeDescription,
                assertionValue  AssertionValue }

        AssertionValue ::= OCTET STRING

        Attribute ::= SEQUENCE {
                type    AttributeDescription,
                vals    SET OF AttributeValue }

        MatchingRuleId ::= LDAPString

        LDAPResult ::= SEQUENCE {
                resultCode      ENUMERATED {
                                success                      (0),
                                operationsError              (1),
                                protocolError                (2),
                                timeLimitExceeded            (3),
                                sizeLimitExceeded            (4),
                                compareFalse                 (5),
                                compareTrue                  (6),
                                authMethodNotSupported       (7),
                                strongAuthRequired           (8),
                                        -- 9 reserved --
                                referral                     (10),  -- new
                                adminLimitExceeded           (11),  -- new
                                unavailableCriticalExtension (12),  -- new
                                confidentialityRequired      (13),  -- new
                                saslBindInProgress           (14),  -- new
                                noSuchAttribute              (16),
                                undefinedAttributeType       (17),
                                inappropriateMatching        (18),
                                constraintViolation          (19),
                                attributeOrValueExists       (20),
                                invalidAttributeSyntax       (21),
                                        -- 22-31 unused --
                                noSuchObject                 (32),
                                aliasProblem                 (33),
                                invalidDNSyntax              (34),
                                -- 35 reserved for undefined isLeaf --
                                aliasDereferencingProblem    (36),
                                        -- 37-47 unused --
                                inappropriateAuthentication  (48),
```

```
                        invalidCredentials          (49),
                        insufficientAccessRights    (50),
                        busy                        (51),
                        unavailable                 (52),
                        unwillingToPerform          (53),
                        loopDetect                  (54),
                                -- 55-63 unused --
                        namingViolation             (64),
                        objectClassViolation        (65),
                        notAllowedOnNonLeaf         (66),
                        notAllowedOnRDN             (67),
                        entryAlreadyExists          (68),
                        objectClassModsProhibited   (69),
                                -- 70 reserved for CLDAP --
                        affectsMultipleDSAs         (71), -- new
                                -- 72-79 unused --
                        other                       (80) },
                        -- 81-90 reserved for APIs --
                matchedDN       LDAPDN,
                errorMessage    LDAPString,
                referral        [3] Referral OPTIONAL }

        Referral ::= SEQUENCE OF LDAPURL

        LDAPURL ::= LDAPString -- limited to characters permitted in URLs

        Controls ::= SEQUENCE OF Control

        Control ::= SEQUENCE {
                controlType             LDAPOID,
                criticality             BOOLEAN DEFAULT FALSE,
                controlValue            OCTET STRING OPTIONAL }

        BindRequest ::= [APPLICATION 0] SEQUENCE {
                version                 INTEGER (1 .. 127),
                name                    LDAPDN,
                authentication          AuthenticationChoice }

        AuthenticationChoice ::= CHOICE {
                simple                  [0] OCTET STRING,
                                         -- 1 and 2 reserved
                sasl                    [3] SaslCredentials }

        SaslCredentials ::= SEQUENCE {
                mechanism               LDAPString,
                credentials             OCTET STRING OPTIONAL }

        BindResponse ::= [APPLICATION 1] SEQUENCE {
```

```
        COMPONENTS OF LDAPResult,
        serverSaslCreds    [7] OCTET STRING OPTIONAL }

    UnbindRequest ::= [APPLICATION 2] NULL

    SearchRequest ::= [APPLICATION 3] SEQUENCE {
            baseObject      LDAPDN,
            scope           ENUMERATED {
                    baseObject              (0),
                    singleLevel             (1),
                    wholeSubtree            (2) },
            derefAliases    ENUMERATED {
                    neverDerefAliases       (0),
                    derefInSearching        (1),
                    derefFindingBaseObj     (2),
                    derefAlways             (3) },
            sizeLimit       INTEGER (0 .. maxInt),
            timeLimit       INTEGER (0 .. maxInt),
            typesOnly       BOOLEAN,
            filter          Filter,
            attributes      AttributeDescriptionList }

    Filter ::= CHOICE {
            and             [0] SET OF Filter,
            or              [1] SET OF Filter,
            not             [2] Filter,
            equalityMatch   [3] AttributeValueAssertion,
            substrings      [4] SubstringFilter,
            greaterOrEqual  [5] AttributeValueAssertion,
            lessOrEqual     [6] AttributeValueAssertion,
            present         [7] AttributeDescription,
            approxMatch     [8] AttributeValueAssertion,
            extensibleMatch [9] MatchingRuleAssertion }

    SubstringFilter ::= SEQUENCE {
            type            AttributeDescription,
            -- at least one must be present
            substrings      SEQUENCE OF CHOICE {
                    initial [0] LDAPString,
                    any     [1] LDAPString,
                    final   [2] LDAPString } }

    MatchingRuleAssertion ::= SEQUENCE {
            matchingRule    [1] MatchingRuleId OPTIONAL,
            type            [2] AttributeDescription OPTIONAL,
            matchValue      [3] AssertionValue,
            dnAttributes    [4] BOOLEAN DEFAULT FALSE }
```



```
    SearchResultEntry ::= [APPLICATION 4] SEQUENCE {
            objectName      LDAPDN,
            attributes      PartialAttributeList }
```

```
PartialAttributeList ::= SEQUENCE OF SEQUENCE {
        type    AttributeDescription,
        vals    SET OF AttributeValue }

SearchResultReference ::= [APPLICATION 19] SEQUENCE OF LDAPURL

SearchResultDone ::= [APPLICATION 5] LDAPResult

ModifyRequest ::= [APPLICATION 6] SEQUENCE {
        object          LDAPDN,
        modification    SEQUENCE OF SEQUENCE {
                operation       ENUMERATED {
                                        add     (0),
                                        delete  (1),
                                        replace (2) },
                modification    AttributeTypeAndValues } }

AttributeTypeAndValues ::= SEQUENCE {
        type    AttributeDescription,
        vals    SET OF AttributeValue }

ModifyResponse ::= [APPLICATION 7] LDAPResult

AddRequest ::= [APPLICATION 8] SEQUENCE {
        entry           LDAPDN,
        attributes      AttributeList }

AttributeList ::= SEQUENCE OF SEQUENCE {
        type    AttributeDescription,
        vals    SET OF AttributeValue }

AddResponse ::= [APPLICATION 9] LDAPResult

DelRequest ::= [APPLICATION 10] LDAPDN

DelResponse ::= [APPLICATION 11] LDAPResult

ModifyDNRequest ::= [APPLICATION 12] SEQUENCE {
        entry           LDAPDN,
        newrdn          RelativeLDAPDN,
        deleteoldrdn    BOOLEAN,
        newSuperior     [0] LDAPDN OPTIONAL }

ModifyDNResponse ::= [APPLICATION 13] LDAPResult
```



```
CompareRequest ::= [APPLICATION 14] SEQUENCE {
        entry           LDAPDN,
        ava             AttributeValueAssertion }

CompareResponse ::= [APPLICATION 15] LDAPResult

AbandonRequest ::= [APPLICATION 16] MessageID
```

```
ExtendedRequest ::= [APPLICATION 23] SEQUENCE {
        requestName      [0] LDAPOID,
        requestValue     [1] OCTET STRING OPTIONAL }

ExtendedResponse ::= [APPLICATION 24] SEQUENCE {
        COMPONENTS OF LDAPResult,
        responseName     [10] LDAPOID OPTIONAL,
        response         [11] OCTET STRING OPTIONAL }

END
```


Full Copyright Statement

Lightweight Directory Access Protocol

From Wikipedia, the free encyclopedia

The **Lightweight Directory Access Protocol**, or **LDAP** (IPA: ['ɛl dæp]), is an application protocol for querying and modifying directory services running over TCP/IP.[1]

A directory is a set of information with similar attributes organized in a logical and hierarchical manner. The most common example is the telephone directory, which consists of a series of names (either of a person or organization) organized alphabetically, with an address and phone number attached.

An LDAP directory tree often reflects various political, geographic, and/or organizational boundaries, depending on the model chosen. LDAP deployments today tend to use Domain Name System (DNS) names for structuring the topmost levels of the hierarchy. Deeper inside the directory might appear entries representing people, organizational units, printers, documents, groups of people or anything else which represents a given tree entry (or multiple entries).

Its current version is LDAPv3, which is specified in a series of Internet Engineering Task Force Standard Track Requests for comments (RFCs) as detailed in RFC 4510 (http://tools.ietf.org/html/rfc4510).

Contents

Origin and influences

Telecommunication companies introduced the concept of directory services to information technology and computer networking, as their understanding of directory requirements was well-developed after some 70 years of producing and managing telephone directories. The culmination of this input was the comprehensive X.500 specification[2], a suite of protocols produced by the International Telecommunication Union (ITU) in the 1980s.

X.500 directory services were traditionally accessed via the X.500 Directory Access Protocol (DAP), which required the Open Systems Interconnection (OSI) protocol stack. LDAP was originally intended to be a "lightweight" alternative protocol for accessing X.500 directory services through the simpler (and now widespread) TCP/IP protocol stack. This model of directory access was borrowed from the DIXIE and Directory Assistance Service protocols.

Standalone LDAP directory servers soon followed, as did directory servers supporting both DAP and LDAP. The latter has become popular in enterprises, as LDAP removed any need to deploy an OSI network. Today, X.500 directory protocols including DAP can also be used directly over TCP/IP.

The protocol was originally created by Tim Howes of the University of Michigan, Steve Kille of ISODE and Wengyik Yeong of Performance Systems International, circa 1993. Further development has been done via the Internet Engineering Task Force (IETF).

In the early engineering stages of LDAP, it was known as *Lightweight Directory Browsing Protocol*, or *LDBP*. It was renamed as the scope of the protocol was expanded to include not only directory browsing and searching functions, but also directory update functions.

LDAP has influenced subsequent Internet protocols, including later versions of X.500, XML Enabled Directory (XED), Directory Service Markup Language (DSML), Service Provisioning Markup Language (SPML), and the Service Location Protocol (SLP).

Protocol overview

A client starts an LDAP session by connecting to an LDAP server, by default on TCP port 389. The client then sends operation requests to the server, and the server sends responses in turn. With some exceptions the client need not wait for a response before sending the next request, and the server may send the responses in any order.

The client may request the following operations:

- Start TLS - optionally protect the connection with Transport Layer Security (TLS), to have a more secure connection
- Bind - authenticate and specify LDAP protocol version
- Search - search for and/or retrieve directory entries
- Compare - test if a named entry contains a given attribute value
- Add a new entry
- Delete an entry
- Modify an entry

- Modify Distinguished Name (DN) - move or rename an entry
- Abandon - abort a previous request
- Extended Operation - generic operation used to define other operations
- Unbind - close the connection (not the inverse of Bind)

In addition the server may send "Unsolicited Notifications" that are not responses to any request, e.g. before it times out a connection.

A common alternate method of securing LDAP communication is using an SSL tunnel. This is denoted in LDAP URLs by using the URL scheme "ldaps". The default port for LDAP over SSL is 636. The use of LDAP over SSL was common in LDAP Version 2 (LDAPv2) but it was never standardized in any formal specification. This usage has been deprecated along with LDAPv2, which was officially retired in 2003 (http://www.ietf.org/IESG/Announcements/draft-zeilenga-ldapv2.ann).

LDAP is defined in terms of ASN.1, and protocol messages are encoded in the binary format BER. It uses textual representations for a number of ASN.1 fields/types, however.

Directory structure

The protocol accesses LDAP directories, which follow the 1993 edition of the X.500 model:

- A directory is a tree of directory entries.
- An entry consists of a set of attributes.
- An attribute has a name (an *attribute type* or *attribute description*) and one or more values. The attributes are defined in a *schema* (see below).
- Each entry has a unique identifier: its *Distinguished Name* (DN). This consists of its *Relative Distinguished Name* (RDN) constructed from some attribute(s) in the entry, followed by the parent entry's DN. Think of the DN as a full filename and the RDN as a relative filename in a folder.

Be aware that a DN may change over the lifetime of the entry, for instance, when entries are moved within a tree. To reliably and unambiguously identify entries, a UUID might be provided in the set of the entry's *operational attributes*.

An entry can look like this when represented in LDIF format (LDAP itself is a binary protocol):

```
dn: cn=John Doe,dc=example,dc=com
cn: John Doe
givenName: John
sn: Doe
telephoneNumber: +1 888 555 6789
telephoneNumber: +1 888 555 1234
mail: john@example.com
manager: cn=Barbara Doe,dc=example,dc=com
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: person
objectClass: top
```

dn is the name of the entry; it's not an attribute nor part of the entry. "cn=John Doe" is the entry's RDN, and "dc=example,dc=com" is the DN of the parent entry. The other lines show the attributes in the entry.

Attribute names are typically mnemonic strings, like "cn" for common name, "dc" for domain component, "mail" for e-mail address and "sn" for surname.

A server holds a subtree starting from a specific entry, e.g. "dc=example,dc=com" and its children. Servers may also hold references to other servers, so an attempt to access "ou=department,dc=example,dc=com" could return a *referral* or *continuation reference* to a server which holds that part of the directory tree. The client can then contact the other server. Some servers also support *chaining*, which means the server contacts the other server and returns the results to the client.

LDAP rarely defines any ordering: The server may return the values in an attribute, the attributes in an entry, and the entries found by a search operation in any order. This follows from the formal definitions - an entry is defined as a set of attributes, and an attribute is a set of values, and sets need not be ordered.

Operations

The client gives each request a positive Message ID, and the server response has the same Message ID. The response includes a numeric result code which indicates success, some error condition or some other special cases. Before the response, the server may send other messages with other result data - for example each entry found by the Search operation is returned in such a message.

> *Expand discussion of referral responses to various operations, especially modify, for example where all modifies must be directed from replicas to a master directory.*

StartTLS

The StartTLS operation establishes Transport Layer Security (the descendant of SSL) on the connection. That can provide data confidentiality (cannot be read by third parties) and/or data integrity protection (protect from tampering). During TLS negotiation the server sends its X.509 certificate to prove its identity. The client may also send a certificate to prove its identity. After doing so, the client may then use SASL/EXTERNAL to have this identity used in determining the identity used in making LDAP authorization decisions.

Servers also often support the non-standard "LDAPS" ("Secure LDAP", commonly known as "LDAP over SSL") protocol on a separate port, by default 636. LDAPS differs from LDAP in two ways: 1) upon connect, the client and server establish TLS before any LDAP messages are transferred (without a Start TLS operation) and 2) the LDAPS connection must be closed upon TLS closure.

LDAPS was primarily used with LDAPv2, because the StartTLS operation had not yet been defined. The use of LDAPS is deprecated, and modern software should only use StartTLS.

Bind (authenticate)

The Bind operation authenticates the client to the server. Simple Bind can send the user's DN and password in plaintext, so the connection should be protected using Transport Layer Security (TLS). The server typically checks the password against the userPassword attribute in the named entry. Anonymous Bind (with empty DN and password) resets the connection to anonymous state. SASL (Simple Authentication and Security Layer) Bind provides authentication services through a wide range of

mechanisms, e.g. Kerberos or the client certificate sent with TLS.

Bind also sets the LDAP protocol version. Normally clients should use LDAPv3, which is the default in the protocol but not always in LDAP libraries.

Bind had to be the first operation in a session in LDAPv2, but is not required in LDAPv3 (the current LDAP version).

Search and Compare

The Search operation is used to both search for and read entries. Its parameters are:

baseObject
: The DN (Distinguished Name) of the entry at which to start the search,

scope
: BaseObject (search just the named entry, typically used to read one entry), singleLevel (entries immediately below the base DN), or wholeSubtree (the entire subtree starting at the base DN).

filter
: How to examine each entry in the scope. E.g. (&(objectClass=person)(|(givenName=John)(mail=john*))) - search for persons who either have given name John or an e-mail address starting with john.

derefAliases
: Whether and how to follow alias entries (entries which refer to other entries),

attributes
: Which attributes to return in result entries.

sizeLimit, timeLimit
: Max number of entries, and max search time.

typesOnly
: Return attribute types only, not attribute values.

The server returns the matching entries and maybe continuation references (in any order), followed by the final result with the result code.

The Compare operation takes a DN, an attribute name and an attribute value, and checks if the named entry contains that attribute with that value.

Update operations

Add, Delete, and Modify DN - all require the DN of the entry that is to be changed.

Modify takes a list of attributes to modify and the modifications to each: Delete the attribute or some values, add new values, or replace the current values with the new ones.

Add operations also can have additional attributes and values for those attributes.

Modify DN (move/rename entry) takes the new RDN (Relative Distinguished Name), optionally the new parent's DN, and a flag which says whether to delete the value(s) in the entry which match the old RDN. The server may support renaming of entire directory subtrees.

An update operation is atomic: Other operations will see either the new entry or the old one. On the

other hand, LDAP does not define transactions of multiple operations: If you read an entry and then modify it, another client may have updated the entry in the mean time. Servers may implement extensions which support this, however.

Extended operations

The Extended Operation is a generic LDAP operation which can be used to define new operations. Examples include the Cancel, Password Modify and Start TLS operations.

Abandon

The Abandon operation requests that the server aborts an operation named by a message ID. The server need not honor the request. Unfortunately, neither Abandon nor a successfully abandoned operation send a response. A similar Cancel extended operation has therefore been defined which does send responses, but not all implementations support this.

Unbind

The Unbind operation abandons any outstanding operations and closes the connection. It has no response. The name is of historical origin: It is *not* the opposite of the Bind operation.

Clients can abort a session by simply closing the connection, but they should use Unbind. Otherwise the server cannot tell the difference between a failed network connection (or a truncation attack) and a discourteous client.

LDAP URLs

An LDAP URL format exists which clients support in varying degree, and which servers return in referrals and continuation references (see RFC 4516 (http://tools.ietf.org/html/rfc4516)):

```
ldap://host:port/DN?attributes?scope?filter?extensions
```

Most of the components, which are described below, are optional.

- *host* is the DNS or IP address of the LDAP server to search.
- *port* is the network port of the LDAP server.
- *DN* is the distinguished name to use as the search base.
- *attributes* is a comma-separated list of attributes to retrieve.
- *scope* specifies the search scope and can be "base" (the default), "one" or "sub".
- *filter* is a search filter, e.g. `(objectClass=*)` (see RFC 4515 (http://tools.ietf.org/html/rfc4515)).
- *extensions* are extensions to the LDAP URL format.

For example, "`ldap://ldap.example.com/cn=John%20Doe,dc=example,dc=com`" refers to all user attributes in John Doe's entry in `ldap.example.com`, while "`ldap:///dc=example,dc=com??sub?(givenName=John)`" searches for the entry in the default server. As in other URLs, special characters must be percent-encoded.

There is a similar non-standard "ldaps:" URL scheme for LDAP over SSL.

Schema

The contents of the entries in a subtree are governed by a schema.

The schema defines the *attribute types* that directory entries can contain. An attribute definition includes a *syntax*, and most non-binary values in LDAPv3 use UTF-8 string syntax. For example, a "mail" attribute might contain the value "user@example.com". A "jpegPhoto" attribute would contain photograph(s) in binary JPEG/JFIF format. A "member" attribute contains DNs of other directory entries. Attribute definitions also specify whether the attribute is single-valued or multi-valued, how to search/compare the attribute (e.g. case-sensitive vs. case-insensitive and whether substring matching is supported), etc.

The schema defines *object classes*. Each entry must have an objectClass attribute, containing named classes defined in the schema. The schema definition of the classes of an entry defines what kind of object the entry may represent - e.g. a person, organization or domain. The object class definitions also list which attributes the entry MAY and MUST contain. For example, an entry representing a person might belong to the classes "top" and "person". Membership in the "person" class would require the entry to contain the "sn" and "cn" attributes, and allow the entry also to contain "userPassword", "telephoneNumber", and other attributes. Since entries may belong to multiple classes, each entry has a complex of optional and mandatory attribute sets formed from the union of the object classes it represents. ObjectClasses can be inherited, and a single entry can have multiple objectClasses to define the available and required attributes of the entry itself. A parallel to the schema of an objectClass is a class definition and an instance in Object-oriented programming, representing LDAP objectClass and LDAP entry, respectively.

The schema also includes various other information controlling directory entries.

Most schema elements have a name and a globally unique Object identifier (OID).

Directory servers may publish the directory schema controlling an entry at a base DN given by the entry's subschemaSubentry operational attribute. (An *operational attribute* describes operation of the directory rather than user information and is only returned from a search when it is explicitly requested.)

Server administrators can define their own schemas in addition to the standard ones. A schema for representing individual people within organizations is termed a white pages schema.

Variations

A lot of the server operation is left to the implementor or administrator to decide. Accordingly, servers may be set up to support a wide variety of scenarios.

For example, data storage in the server is not specified - the server may use flat files, databases, or just be a gateway to some other server. Access control is not standardized, though there has been work on it and there are commonly used models. Users' passwords may be stored in their entries or elsewhere. The server may refuse to perform operations when it wishes, and impose various limits.

Most parts of LDAP are extensible. Examples: One can define new operations. *Controls* may modify requests and responses, e.g. to request sorted search results. New search scopes and Bind methods can be defined. Attributes can have *options* that may modify their semantics..

Other data models

As LDAP has gained momentum, vendors have provided it as an access protocol to other services. The implementation then recasts the data to mimic the LDAP/X.500 model, but how closely this model is followed varies. For example, there is software to access SQL databases through LDAP, even though LDAP does not readily lend itself to this. X.500 servers may support LDAP as well.

Similarly, data which were previously held in other types of data stores are sometimes moved to LDAP directories. For example, Unix user and group information can be stored in LDAP and accessed via PAM and NSS modules. LDAP is often used by other services for authentication.

Usage

Applications

One reason to choose LDAP for a service is that it is quite widely supported, and data presented in LDAP is thus immediately available to many clients and libraries. Another is that LDAP is very general and includes basic security, and can support many types of applications.

Thus, if one chooses a few general protocols like LDAP and HTTP for various services, one can focus on these few protocols instead of having to maintain and upgrade many specialized protocols.

Two common applications of LDAP are for computer user/group data, and for address book information (persons, departments etc). Many e-mail clients support LDAP lookups.

Some tasks LDAP does not handle well are to model a relational database, and data whose ordering must be preserved. (Though an extension does exist for the latter.)

Naming structure

Since an LDAP server can return referrals to other servers for requests the server itself will not/can not serve, a naming structure for LDAP entries is needed so one can find a server holding a given DN. Since such a structure already exists in the Domain name system (DNS), servers' top level names often mimic DNS names.

If an organization has domain name foo.example, its top level LDAP entry will therefore typically have the DN dc=foo,dc=example (where dc means domain component). If the ldap server is also named ldap.foo.example, the organization's top level LDAP URL becomes ldap://ldap.foo.example/dc=foo,dc=example.

Below the top level, the entry names will typically reflect the organization's internal structure or needs rather than DNS names.

This is also a descendent of the X.500 series

Terminology

The LDAP terminology one can encounter is rather cumbersome. Some of this is due to misunderstandings, other examples are due to its historical origins, others arise when used with non-X.500 services that use different terminology. For example, "LDAP" is sometimes used to refer to the protocol, other times to the protocol and the data. An "LDAP directory" may be the data or also the access point. An "attribute" may be the attribute type, or the contents of an attribute in a directory, or an attribute description (an attribute type with *options*). An "anonymous" and an "unauthenticated" Bind are different Bind methods that both produce anonymous authentication state, so both terms are being used for both variants. The "uid" attribute should hold user names rather than numeric user IDs.

Browsers

- Apache Directory Studio (http://directory.apache.org/studio/): an LDAP browser and directory client for Linux, MacOS and Windows, and as a plug-in for the Eclipse development environment.
- JXplorer - A Java-based browser that runs in any operating environment
- LDAP Browser (http://www.ldapbrowser.com/download.htm): a Windows-only browser
- LDAPAdmin (http://ldapadmin.sourceforge.net/): a Windows-only browser
- MaXware Directory Explorer (http://www.maxware.com/products/mde.html): a Windows-only browser
- OpenLDAP
- LDAP Browser/Editor (http://www-unix.mcs.anl.gov/~gawor/ldap/) - A Java-based LDAP browser / editor

References

1. ^ LDAP: Framework, Practices, and Trends (http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/mags/ic/&toc=comp/mags/ic/2004/05/w5toc.xml&DOI=10.1109/MIC.2004.44)
2. ^ The X.500 series - ITU-T Rec. X.500 to X.521

- ITU-T Rec. X.680, "Abstract Syntax Notation One (ASN.1) - Specification of Basic Notation", 1994
- Basic encoding rules (BER) - ITU-T Rec. X.690, "Specification of ASN.1 encoding rules: Basic, Canonical, and Distinguished Encoding Rules", 1994
- RFC 4346 (http://tools.ietf.org/html/rfc4346) - The TLS Protocol Version 1.1
- RFC 4422 (http://tools.ietf.org/html/rfc4422) - Simple Authentication and Security Layer (SASL)
- SASL mechanisms (http://www.iana.org/assignments/sasl-mechanisms) registered at IANA

See also

- OpenDS
- OpenLDAP
- Fedora Directory Server

- Apache Directory Server
- Sun Java System Directory Server
- Apple Open Directory Server
- Directory service
- LDAP Data Interchange Format (LDIF)
- LDAP Application Program Interface
- Key server (cryptographic)
- Simple Authentication and Security Layer (SASL)
- Transport Layer Security (TLS)
- X.500

External links

- LDAP Authentication (for Linux/Unix and Windows via Samba) (http://linuxwiki.riverworth.com/index.php/LDAP_Authentication)
- PALDAP, A Lazy Directory Administrator's Pal (http://www.paldap.org/) An LDAP Wiki
- Linux LDAP HOWTO (http://www.tldp.org/HOWTO/LDAP-HOWTO/)
- LDAP Articles, Links, Whitepapers (http://www.bind9.net/ldap/)
- LDAP Software, Tools & Utilities (http://www.bind9.net/ldap-tools)
- SIDVault - Windows LDAP Server (http://alphacentauri.co.nz/sidvault/index.htm)
- LDAP Libraries for C# (http://forge.novell.com/modules/xfmod/project/showfiles.php?group_id=1318).
- LDAP for Rocket Scientists (http://www.zytrax.com/books/ldap/) - Comprehensive LDAP Guide/Tutorial
- The importance of LDAP (http://www.freesoftwaremagazine.com/free_issues/issue_03/ldap/) A commentary by Tom Jackiewicz about LDAP
- A Wiki for using LDAP in Debian (http://wiki.debian.org/LDAP)
- A loose collection of Linux LDAP tutorials (http://yolinux.com/TUTORIALS/LinuxTutorialLDAP.html)
- OneLDAP (http://www.nabber.org/projects/oneldap/) An LDAP gateway to standard protocols like POP3 and IMAP.
- LDAP schema design (http://www.skills-1st.co.uk/papers/ldap-schema-design-feb-2005/index.html)
- Security with LDAP (http://www.skills-1st.co.uk/papers/security-with-ldap-jan-2002/index.html)
- Understanding LDAP (http://www.devshed.com/c/a/Administration/Understanding-LDAP-part-1/) A simple, light introductory tutorial for LDAP.
- OpenDS (http://www.opends.org/) An open source directory server, written in Java

LDAP forums

- LDAP mailinglist at umich.edu (http://www.umich.edu/~dirsvcs/ldap/mailinglist.html)
- "#ldap" IRC channel in the freenode IRC network
- IETF LDAPbis mailing list (http://www.openldap.org/lists/mm/listinfo/ietf-ldapbis) - The IETF LDAP (v3) Revision (LDAPbis) (http://www.ietf.org/html.charters/OLD/ldapbis-charter.html) Working Group, now concluded, was chartered with revising the LDAPv3 specifications. The WG produced a number of RFCs, including the revised LDAP Technical Specification detailed in RFC 4510 (http://tools.ietf.org/html/rfc4510). The WG mailing list remains available for technical discussions regarding the LDAP Technical Specification.
- IETF LDAPext mailing list (https://www1.ietf.org/mailman/listinfo/ldapext) - The IETF LDAP

Extensions (LDAPext) (http://www.ietf.org/html.charters/OLD/ldapext-charter.html) Working Group, now concluded, was charted with producing a set of LDAP extensions. The WG mailing list remains available for technical discussions regarding the LDAP extension specifications.

RFCs

LDAP is currently specified in a series of Request for Comments documents:

- RFC 4510 (http://tools.ietf.org/html/rfc4510) - Lightweight Directory Access Protocol (LDAP) Technical Specification Roadmap (replaced the previous LDAP Technical specification, RFC 3377 (http://tools.ietf.org/html/rfc3377), in its entirety)
- RFC 4511 (http://tools.ietf.org/html/rfc4511) - Lightweight Directory Access Protocol (LDAP): The Protocol
- RFC 4512 (http://tools.ietf.org/html/rfc4512) - Lightweight Directory Access Protocol (LDAP): Directory Information Models
- RFC 4513 (http://tools.ietf.org/html/rfc4513) - Lightweight Directory Access Protocol (LDAP): Authentication Methods and Security Mechanisms
- RFC 4514 (http://tools.ietf.org/html/rfc4514) - Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names
- RFC 4515 (http://tools.ietf.org/html/rfc4515) - Lightweight Directory Access Protocol (LDAP): String Representation of Search Filters
- RFC 4516 (http://tools.ietf.org/html/rfc4516) - Lightweight Directory Access Protocol (LDAP): Uniform Resource Locator
- RFC 4517 (http://tools.ietf.org/html/rfc4517) - Lightweight Directory Access Protocol (LDAP): Syntaxes and Matching Rules
- RFC 4518 (http://tools.ietf.org/html/rfc4518) - Lightweight Directory Access Protocol (LDAP): Internationalized String Preparation
- RFC 4519 (http://tools.ietf.org/html/rfc4519) - Lightweight Directory Access Protocol (LDAP): Schema for User Applications

The following RFCs detail LDAP-specific Best Current Practices:

- RFC 4520 (http://tools.ietf.org/html/rfc4520) (also BCP 64) - Internet Assigned Numbers Authority (IANA) Considerations for the Lightweight Directory Access Protocol (LDAP) (replaced RFC 3383 (http://tools.ietf.org/html/rfc3383))
- RFC 4521 (http://tools.ietf.org/html/rfc4521) (also BCP 118) - Considerations for Lightweight Directory Access Protocol (LDAP) Extensions

The following is a partial list of RFCs specifying LDAPv3 extensions:

- RFC 2247 (http://tools.ietf.org/html/rfc2247) - Use of DNS domains in distinguished names
- RFC 2307 (http://tools.ietf.org/html/rfc2307) - Using LDAP as a Network Information Service
- RFC 2589 (http://tools.ietf.org/html/rfc2589) - LDAPv3: Dynamic Directory Services Extensions
- RFC 2649 (http://tools.ietf.org/html/rfc2649) - LDAPv3 Operational Signatures
- RFC 2696 (http://tools.ietf.org/html/rfc2696) - LDAP Simple Paged Result Control
- RFC 2798 (http://tools.ietf.org/html/rfc2798) - inetOrgPerson LDAP Object Class
- RFC 2849 (http://tools.ietf.org/html/rfc2849) - The LDAP Data Interchange Format (LDIF)
- RFC 2891 (http://tools.ietf.org/html/rfc2891) - Server Side Sorting of Search Results
- RFC 3045 (http://tools.ietf.org/html/rfc3045) - Storing Vendor Information in the LDAP root DSE
- RFC 3062 (http://tools.ietf.org/html/rfc3062) - LDAP Password Modify Extended Operation

- RFC 3296 (http://tools.ietf.org/html/rfc3296) - Named Subordinate References in LDAP Directories
- RFC 3671 (http://tools.ietf.org/html/rfc3671) - Collective Attributes in LDAP
- RFC 3672 (http://tools.ietf.org/html/rfc3672) - Subentries in LDAP
- RFC 3673 (http://tools.ietf.org/html/rfc3673) - LDAPv3: All Operational Attributes
- RFC 3687 (http://tools.ietf.org/html/rfc3687) - LDAP Component Matching Rules
- RFC 3698 (http://tools.ietf.org/html/rfc3698) - LDAP: Additional Matching Rules
- RFC 3829 (http://tools.ietf.org/html/rfc3829) - LDAP Authorization Identity Controls
- RFC 3866 (http://tools.ietf.org/html/rfc3866) - Language Tags and Ranges in LDAP
- RFC 3909 (http://tools.ietf.org/html/rfc3909) - LDAP Cancel Operation
- RFC 3928 (http://tools.ietf.org/html/rfc3928) - LDAP Client Update Protocol
- RFC 4370 (http://tools.ietf.org/html/rfc4370) - LDAP Proxied Authorization Control
- RFC 4373 (http://tools.ietf.org/html/rfc4373) - LBURP
- RFC 4403 (http://tools.ietf.org/html/rfc4403) - LDAP Schema for UDDI
- RFC 4522 (http://tools.ietf.org/html/rfc4522) - LDAP: Binary Encoding Option
- RFC 4523 (http://tools.ietf.org/html/rfc4523) - LDAP: X.509 Certificate Schema
- RFC 4524 (http://tools.ietf.org/html/rfc4524) - LDAP: COSINE Schema (replaces RFC 1274 (http://tools.ietf.org/html/rfc1274))
- RFC 4525 (http://tools.ietf.org/html/rfc4525) - LDAP: Modify-Increment Extension
- RFC 4526 (http://tools.ietf.org/html/rfc4526) - LDAP: Absolute True and False Filters
- RFC 4527 (http://tools.ietf.org/html/rfc4527) - LDAP: Read Entry Controls
- RFC 4528 (http://tools.ietf.org/html/rfc4528) - LDAP: Assertion Control
- RFC 4529 (http://tools.ietf.org/html/rfc4529) - LDAP: Requesting Attributes by Object Class
- RFC 4530 (http://tools.ietf.org/html/rfc4530) - LDAP: entryUUID
- RFC 4531 (http://tools.ietf.org/html/rfc4531) - LDAP Turn Operation
- RFC 4532 (http://tools.ietf.org/html/rfc4532) - LDAP Who am I? Operation
- RFC 4533 (http://tools.ietf.org/html/rfc4533) - LDAP Content Sync Operation

LDAPv2 was specified in the following RFCs:

- RFC 1777 (http://tools.ietf.org/html/rfc1777) - Lightweight Directory Access Protocol (replaced RFC 1487 (http://tools.ietf.org/html/rfc1487))
- RFC 1778 (http://tools.ietf.org/html/rfc1778) - The String Representation of Standard Attribute Syntaxes (replaced RFC 1488 (http://tools.ietf.org/html/rfc1488))
- RFC 1779 (http://tools.ietf.org/html/rfc1779) - A String Representation of Distinguished Names (replaced RFC 1485 (http://tools.ietf.org/html/rfc1485))

LDAPv2 was moved to historic status by the following RFC:

- RFC 3494 (http://tools.ietf.org/html/rfc3494) - Lightweight Directory Access Protocol version 2 (LDAPv2) to Historic Status

Retrieved from "http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol"

Categories: Articles with sections needing expansion | Internet protocols | Internet standards | Identity management | Application layer protocols

Created by Gracion

ClickMail LDAP

DigiTunnel VPN

DoorStop Firewall

LDAP Resources:

LDAP Books

Wikipedia

Intro to LDAP

LDAP roadmap

RFC's

X.500 standard

Schemas:
inetOrgPerson
IWPS (I'net White Pages)

Netscape LDAP FAQ

Netscape Attribute Dictionary

OpenLDAP

ClickMail Tour

Admin's view (entry window)

User's view:
OS X
Outlook
Netscape,
QM Pro
Eudora

AppleShare IP

Gracion Software

What is LDAP?

LDAP, Lightweight Directory Access Protocol, is an Internet protocol that email and other programs use to look up information from a server.

Every email program has a personal address book, but how do you look up an address for someone who's never sent you email? How can an organization keep one centralized up-to-date phone book that everybody has access to?

That question led software companies such as Microsoft, IBM, Lotus, and Netscape to support a standard called LDAP. "LDAP-aware" client programs can ask LDAP servers to look up entries in a wide variety of ways. LDAP servers index all the data in their entries, and "filters" may be used to select just the person or group you want, and return just the information you want. For example, here's an LDAP search translated into plain English: "Search for all people located in Chicago whose name contains "Fred" that have an email address. Please return their full name, email, title, and description."

LDAP is not limited to contact information, or even information about people. LDAP is used to look up encryption certificates, pointers to printers and other services on a network, and provide "single signon" where one password for a user is shared between many services. LDAP is appropriate for any kind of directory-like information, where fast lookups and less-frequent updates are the norm.

As a protocol, LDAP does not define how programs work on either the client or server side. It defines the "language" used for client programs to talk to servers (and servers to servers, too). On the client side, a client may be an email program, a printer browser, or an address book. The server may speak only LDAP, or

integration

What is LDAP?

Specifications

Press Info

have other methods of sending and receiving data—LDAP may just be an add-on method.

If you have an email program (as opposed to web-based email), it probably supports LDAP. Most LDAP clients can only read from a server. Search abilities of clients (as seen in email programs) vary widely. A few can write or update information, but LDAP does not include security or encryption, so updates usually requre additional protection such as an encrypted SSL connection to the LDAP server.

LDAP also defines: **Permissions,** set by the administrator to allow only certain people to access the LDAP database, and optionally keep certain data private. **Schema:** a way to describe the format and attributes of data in the server. For example: a schema entered in an LDAP server might define a "groovyPerson" entry type, which has attributes of "instantMessageAddress", and "coffeeRoastPreference". The normal attributes of name, email address, etc., would be inherited from one of the standard schemas, which are rooted in X.500 (see below).

LDAP was designed at the University of Michigan to adapt a complex enterprise directory system (called X.500) to the modern Internet. X.500 is too complex to support on desktops and over the Internet, so LDAP was created to provide this service "for the rest of us."

LDAP servers exist at three levels: There are big public servers, large organizational servers at universities and corporations, and smaller LDAP servers for workgroups. Most public servers from around year 2000 have disappeared, although directory.verisign.com exists for looking up X.509 certificates. The idea of publicly listing your email address for the world to see, of course, has been crushed by spam.

While LDAP didn't bring us the worldwide email address book, it continues to be a popular standard for communicating record-based, directory-like data between programs.

We hope we've helped answer your question, "What is LDAP?" For more information use the links at left. For LDAP product information use the Google ad links at right.

This page brought to you by the makers of DigiTunnel VPN Client for Mac OS X and ClickMail Central Directory, the LDAP server for classic Macintosh systems. Clickmail is still available but no longer under developement.

To other pages in our guided tour of ClickMail Central Directory:

previous continued

An Introduction to LDAP

Michael Donnelly

If you work in the computing industry, the chances are good that you've heard of LDAP by now. Wondering what all the excitement is about? Want to know a little more about the underlying technology? You've come to the right place. This introduction - the first in a series of articles describing how to design, implement, and integrate an LDAP environment at your company - will familiarize you with the concepts behind LDAP while leaving the really hardcore details for later. Here, we'll touch on the following topics:

To start with, what's happening with LDAP today *is* exciting. A company-wide LDAP implementation can enable almost any application, running on almost any computer platform, to obtain information from your LDAP directory. And that directory can be used to store a broad range of data: email address and mail routing information, HR data, public security keys, contact lists, and much more. By making an LDAP directory a focal point in your systems integration, you're providing one-stop shopping whenever people go looking for information within your company - even if the primary source of the data lives elsewhere.

But wait, you say. You're already using an Oracle, Sybase, Informix, or Microsoft SQL database to store much of that same data. How is LDAP different? What makes it better? Read on.

What is LDAP, anyway?

The Lightweight Directory Access Protocol, better known as LDAP, is based on the X.500 standard, but significantly simpler and more readily adapted to meet custom needs. Unlike X.500, LDAP supports TCP/IP, which is necessary for Internet access. The core LDAP specifications are all defined in RFCs -- a complete list of LDAP-related RFCs may be found at the LDAPman RFC page.

Using "LDAP" in a sentence
In everyday conversation, you'll hear well-intentioned people say things like, "Should we be storing that in LDAP?" or "Just get that data from the LDAP database," or "How do we go about tying LDAP into an RDB?" Strictly speaking, though, LDAP isn't a database at all, but a protocol used to access information stored in an information directory (also known as an LDAP directory). A more precise formulation might look something like this: "Using LDAP, data will be retrieved from (or stored in) the correct location within our information directory." But you won't find me correcting anyone on this point: either way, you get the idea across, and that's what counts.

Is an LDAP information directory a database?
Just as a Database Management System (DBMS) from Sybase, Oracle, Informix, or Microsoft is used to

process queries and updates to a relational database, an LDAP server is used to process queries and updates to an LDAP information directory. In other words, an LDAP information directory *is* a type of database, but it's *not* a relational database. And unlike databases that are designed for processing hundreds or thousands of changes per minute - such as the Online Transaction Processing (OLTP) systems often used in e-commerce - LDAP directories are heavily optimized for read performance.

The advantages of LDAP directories

Now that we've straightened that out, what are the advantages of LDAP directories? The current popularity of LDAP is the culmination of a number of factors. I'll give you a few basic reasons, provided you keep in mind that it's just part of the story.

Perhaps the biggest plus for LDAP is that your company can access the LDAP directory from almost any computing platform, from any one of the increasing number of readily available, LDAP-aware applications. It's also easy to customize your company's internal applications to add LDAP support.

The LDAP protocol is both cross-platform and standards-based, so applications needn't worry about the type of server hosting the directory. In fact, LDAP is finding much wider industry acceptance because of its status as an Internet standard. Vendors are more willing to write LDAP integration into their products because they don't have to worry about what's at the other end. Your LDAP server could be any one of a number of open-source or commercial LDAP directory servers (or perhaps even a DBMS server with an LDAP interface), since interacting with any true LDAP server involves the same protocol, client connection package, and query commands. By contrast, vendors looking to integrate directly with a DBMS usually must tailor their product to work with each database server vendor individually.

Unlike many relational databases, you do not have to pay for either client connection software or for licensing.

Most LDAP servers are simple to install, easily maintained, and easily optimized.

LDAP servers can replicate either some or all of their data via push or pull methods, allowing you to push data to remote offices, to increase security, and so on. The replication technology is built-in and easy to configure. By contrast, many of the big DBMS vendors charge extra for this feature, and it's far more difficult to manage.

LDAP allows you to securely delegate read and modification authority based on your specific needs using ACIs (collectively, an ACL, or Access Control List). For example, your facilities group might be given access to change an employee's location, cube, or office number, but not be allowed to modify entries for any other fields. ACIs can control access depending on who is asking for the data, what data is being asked for, where the data is stored, and other aspects of the record being modified. This is all done through the LDAP directory directly, so you needn't worry about making security checks at the user application level.

LDAP is particularly useful for storing information that you wish to read from many locations, but update infrequently. For example, your company could store all of the following very efficiently in an LDAP directory:

- The company employee phone book and organizational chart
- External customer contact information
- Infrastructure services information, including NIS maps, email aliases, and so on
- Configuration information for distributed software packages

- Public certificates and security keys

When should you use LDAP to store your data?

Most LDAP servers are heavily optimized for read-intensive operations. Because of this, one can typically see an order of magnitude difference when reading data from an LDAP directory versus obtaining the same data from a relational database server optimized for OLTP. Because of this optimization, however, most LDAP directories are not well suited for storing data where changes are frequent. For instance, an LDAP directory server is great for storing your company's internal telephone directory, but don't even think of using it as a database back end for your high-volume e-commerce site.

If the answer to each of the following questions is Yes, then storing your data in LDAP is a good idea.

- Would you like your data to be available cross-platform?
- Do you need to access this data from a number of computers or applications?
- Do the individual records you're storing change a few times a day or less, on average?
- Does it make sense to store this type of data in a flat database instead of a relational database? That is, could you effectively store all the data for a given item in a single record?

This final question often gives people pause, because it's very common to access a flat record to obtain data that's relational in nature. For example, a record for a company employee might include the login name of that employee's manager. It's fine to use LDAP to store this kind of information. Rule of thumb: If you can imagine storing your data in a large electronic Rolodex, you can store it easily in an LDAP directory.

The structure of an LDAP directory tree

LDAP directory servers store their data hierarchically. If you've seen the top-down representations of DNS trees or UNIX file directories, an LDAP directory structure will be familiar ground. As with DNS host names, an LDAP directory record's Distinguished Name (DN for short) is read from the individual entry, backwards through the tree, up to the top level. More on this point later.

Why break things up into a hierarchy? There are a number of reasons. Here are a few possible scenarios:

- You may wish to push all your US-based customer contact information to an LDAP server in the Seattle office (which is devoted to sales), whereas you probably don't need to push the company's asset management information there.
- You may wish to grant permissions to a group of individuals based on the directory structure. In the example listed below, the company's asset management team might need full access to the asset-mgmt section, but not to other areas.
- Combined with replication, you can tailor the layout of your directory structure to minimize WAN bandwidth utilization. Your sales office in Seattle might need up-to-the minute updates for US sales contacts, but only hourly updates for European sales information.

Getting to the root of the matter: Your base DN and you
The top level of the LDAP directory tree is the base, referred to as the "base DN." A base DN usually takes one of the three forms listed here. Let's assume I work at a US electronic commerce company called FooBar, Inc., which is on the Internet at foobar.com.

o="FooBar, Inc.", c=US
(base DN in X.500 format)
In this example, `o=FooBar, Inc.` refers to the organization, which in this context should be treated as synonymous with the company name. `c=US` indicates that the company headquarters is in the US. Once upon a time, this was the preferred method of specifying your base DN. Times and fashions change, though; these days, most companies are (or plan to be) on the Internet. And what with Internet globalization, using a country code in the base DN probably made things more confusing in the end. In time, the X.500 format evolved into the other formats listed below.

o=foobar.com
(base DN derived from the company's Internet presence)
This format is fairly straightforward, using the company's Internet domain name as the base. Once you get past the `o=` portion (which stands for `organization=`), everyone at your company should know where the rest came from. This was, until recently, probably the most common of the currently used formats.

dc=foobar, dc=com
(base DN derived from the company's DNS domain components)
As with the previous format, this uses the DNS domain name as its basis. But where the other format leaves the domain name intact (and thus human-readable), this format is split into domain components: `foobar.com` becomes `dc=foobar, dc=com`. In theory, this could be slightly more versatile, though it's a little harder for end users to remember. By way of illustration, consider foobar.com. When foobar.com merges with gizmo.com, you simply start thinking of "dc=com" as the base DN. Place the new records into your existing directory under dc=gizmo, dc=com, and you're ready to go. (Of course, this approach doesn't help if foobar.com merges with wocket.edu.) This is the format I'd recommend for any new installations. Oh, and if you're planning to use Active Directory, Microsoft has already decided for you that this is the format you wanted.

Time to branch out: How to organize your data in your directory tree
In a UNIX file system, the top level is the root. Beneath the root you have numerous files and directories. As mentioned above, LDAP directories are set up in much the same manner.

Underneath your directory's base, you'll want to create containers that logically separate your data. For historical (X.500) reasons, most LDAP directories set these logical separations up as OU entries. OU stands for "Organizational Unit," which in X.500 was used to indicate the functional organization within a company: sales, finance, et cetera. Current LDAP implementations have kept the `ou=` naming convention, but break things apart by broad categories like `ou=people`, `ou=groups`, `ou=devices`, and so on. Lower level OUs are sometimes used to break categories down further. For example, an LDAP directory tree (not including individual entries) might look like this:

```
dc=foobar, dc=com
    ou=customers
        ou=asia
        ou=europe
        ou=usa
    ou=employees
    ou=rooms
    ou=groups
    ou=assets-mgmt
    ou=nisgroups
    ou=recipes
```

Individual LDAP records

What's in a name? The DN of an LDAP entry
All entries stored in an LDAP directory have a unique "Distinguished Name," or DN. The DN for each LDAP entry is composed of two parts: the Relative Distinguished Name (RDN) and the location within the LDAP directory where the record resides.

The RDN is the portion of your DN that is not related to the directory tree structure. Most items that you'll store in an LDAP directory will have a name, and the name is frequently stored in the `cn` (Common Name) attribute. Since nearly everything has a name, most objects you'll store in LDAP will use their `cn` value as the basis for their RDN. If I'm storing a record for my favorite oatmeal recipe, I'll be using `cn=Oatmeal Deluxe` as the RDN of my entry.

- My directory's base DN is `dc=foobar,dc=com`
- I'm storing all the LDAP records for my recipes in `ou=recipes`
- The RDN of my LDAP record is `cn=Oatmeal Deluxe`

Given all this, what's the full DN of the LDAP record for this oatmeal recipe? Remember, it reads backwards - just like a host name in DNS.

cn=Oatmeal Deluxe,ou=recipes,dc=foobar,dc=com

People are always more trouble than inanimate objects
Now it's time to tackle the DN of a company employee. For user accounts, you'll typically see a DN based either on the `cn` or on the `uid` (User ID). For example, the DN for FooBar's employee Fran Smith (login name: fsmith) might look like either of these two formats:

uid=fsmith,ou=employees,dc=foobar,dc=com
(login-based)
LDAP (and X.500) use `uid` to mean "User ID", not to be confused with the UNIX uid number. Most companies try to give everyone a unique login name, so this approach makes good sense for storing information about employees. You don't have to worry about what you'll do when you hire the next Fran Smith, and if Fran changes her name (marriage? divorce? religious experience?), you won't have to change the DN of the LDAP entry.

cn=Fran Smith,ou=employees,dc=foobar,dc=com
(name-based)
Here we see the Common Name (CN) entry used. In the case of an LDAP record for a person, think of the common name as their full name. One can easily see the downside to this approach: if the name changes, the LDAP record has to "move" from one DN to another. As indicated above, you want to avoid changing the DN of an entry whenever possible.

Customizing your directory's object classes

You can use LDAP to store data on almost any type of object, as long as that object can be described in terms of various attributes. Here are a few examples of information you might store:

- Employees: What's the employee's full name, login name, password, employee number, manager's login, mail server?

- Asset tracking: What's the computer name, IP address, asset tag, make and model information, physical location?
- Customer contact lists: What's the customer's company name? The primary contact's phone, fax, and email information?
- Meeting room information: What's the room name, location, seating capacity, telephone number? Is there wheelchair access? Is there an overhead projector?
- Recipe information: Give the name of the dish, the list of ingredients, the type of cuisine, and instructions for preparing it.

Because your LDAP directory can be customized to store any type of text or binary data, what you store is really up to you. LDAP directories use the concept of object classes to define which attributes are allowed for objects of any given type. In almost every LDAP implementation, you'll want to extend the basic functionality of your LDAP directory to meet your specific needs, either by creating new object classes or by extending existing ones.

LDAP directories store all information for a given record's entries as a series of attribute pairs, each one consisting of an attribute type and an attribute value. (This is completely different from the way relational database servers store data, in columns and rows.) Consider this portion of my recipe record, as stored in an LDAP directory:

```
dn: cn=Oatmeal Deluxe, ou=recipes, dc=foobar, dc=com
cn: Instant Oatmeal Deluxe
recipeCuisine: breakfast
recipeIngredient: 1 packet instant oatmeal
recipeIngredient: 1 cup water
recipeIngredient: 1 pinch salt
recipeIngredient: 1 tsp brown sugar
recipeIngredient: 1/4 apple, any type
```

Note that in this case, each ingredient is listed as a value of attribute type `recipeIngredient`. LDAP directories are designed to store multiple values of a single type in this fashion, rather than storing the entire list in a single database field with some sort of delimiter to distinguish the individual values.

Because the data is stored in this way, the shape of the database can be completely fluid - you don't need to recreate a database table (and all its indexes) to start tracking a new piece of data. Even more important, LDAP directories use no memory or storage to handle "empty" fields - in fact, having unused optional fields costs you nothing at all.

An example of an individual LDAP entry

Let's look at an example. We'll use the LDAP record of Fran Smith, our friendly employee from Foobar, Inc. The format of this entry is LDIF, the format used when exporting and importing LDAP directory entries.

```
dn: uid=fsmith, ou=employees, dc=foobar, dc=com
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
objectclass: foobarPerson
uid: fsmith
givenname: Fran
sn: Smith
```

```
cn: Fran Smith
cn: Frances Smith
telephonenumber: 510-555-1234
roomnumber: 122G
o: Foobar, Inc.
mailRoutingAddress: fsmith@foobar.com
mailhost: mail.foobar.com
userpassword: {crypt}3x1231v76T89N
uidnumber: 1234
gidnumber: 1200
homedirectory: /home/fsmith
loginshell: /usr/local/bin/bash
```

To start with, attribute values are stored with case intact, but searches against them are case-insensitive by default. Certain attributes (like password) are case-sensitive when searching.

Let's break this entry down and look at it piece by piece.

```
dn: uid=fsmith, ou=employees, dc=foobar, dc=com
```

This is the full DN of Fran's LDAP entry, including the whole path to the entry in the directory tree. LDAP (and X.500) use `uid` to mean "User ID," not to be confused with the UNIX uid number.

```
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
objectclass: foobarPerson
```

One can assign as many object classes as are applicable to any given type of object. The `person` object class requires that the `cn` (common name) and `sn` (surname) fields have values. Object Class `person` also allows other optional fields, including givenname, telephonenumber, and so on. The object class `organizationalPerson` adds more options to the values from `person`, and `inetOrgPerson` adds still more options to that (including email information). Finally, `foobarPerson` is Foobar's customized object class that adds all the custom attributes they wish to track at their company.

```
uid: fsmith
givenname: Fran
sn: Smith
cn: Fran Smith
cn: Frances Smith
telephonenumber: 510-555-1234
roomnumber: 122G
o: Foobar, Inc.
```

As mentioned before, `uid` stands for User ID. Just translate it in your head to "login" whenever you see it.

Note that there are multiple entries for the CN. As mentioned above, LDAP allows some attributes to have multiple values, with the number of values being arbitrary. When would you want this? Let's say you're searching the company LDAP directory for Fran's phone number. While *you* might know her as Fran (having heard her spill her guts over lunchtime margaritas on more than one occasion), the people in HR may refer to her (somewhat more formally) as Frances. Because both versions of her name are stored, either search will successfully look up Fran's telephone number, email, cube number, and so on.

```
mailRoutingAddress: fsmith@foobar.com
mailhost: mail.foobar.com
```

Like most companies on the Internet, Foobar uses Sendmail for internal mail delivery and routing. Foobar stores all users' mail routing information in LDAP, which is fully supported by recent versions of Sendmail.

```
userpassword: {crypt}3x1231v76T89N
 uidnumber: 1234
 gidnumber: 1200
 gecos: Frances Smith
 homedirectory: /home/fsmith
 loginshell: /usr/local/bin/bash
```

Note that Foobar's systems administrators store all the NIS password map information in LDAP as well. At Foobar, the `foobarPerson` object class adds this capability. Note that the user password is stored in UNIX crypt format. The UNIX uid is stored here as `uidnumber`. Mind you, there's a whole RFC on storing NIS information in LDAP. I'll talk about NIS integration in a future article.

LDAP replication

LDAP servers can be set to replicate some or all of their data, on a push or a pull basis, using simple authentication or certificate-based authentication.

For example, Foobar has a "public" LDAP server running on ldap.foobar.com, port 389. This server is used by Netscape Communicator's pinpoint email addressing feature, the "ph" command from UNIX, and other locations where a user would want to query for the phone number of an employee or customer contact. The company's master LDAP server is running on the same system, but on port 1389 instead.

You wouldn't necessarily want employees searching the directory to query against asset management or recipe data, nor would it be desirable to see IT accounts (like "root") showing up on the company directory. To accomodate these unpleasant realities, Foobar replicates selected directory subtrees from its master LDAP server to its "public" server. The replication excludes subtrees containing data they wish to hide. To keep things current at all times, the master directory server is set to do immediate push-based synchronization. Note that this approach is designed for convenience, not security: the idea is to allow power users to simply query the other LDAP port if they want to search all available data.

Let's say Foobar is managing its customer contact information via LDAP, over a low bandwidth connection between Oakland and Europe. They might set up replication from ldap.foobar.com:1389 to munich-ldap.foobar.com:389 as follows:

periodic pull: `ou=asia,ou=customers,o=sendmail.com`
periodic pull: `ou=us,ou=customers,o=sendmail.com`
immediate push: `ou=europe,ou=customers,o=sendmail.com`

The pull connections would keep things in sync every 15 minutes, which would probably be just fine in this scenario. The push connection would guarantee that any change made to the European contact information would be pushed out to Munich immediately.

Given this replication scheme, where would users connect to access their data? Users in Munich could simply connect to their local server. If they were making changes to the data, the local LDAP server

would refer those changes to the master LDAP server, which would then push all the changes back to the local LDAP server to keep it in sync. This is of tremendous benefit to the local user: all their LDAP queries (mostly reads) are against their local server, which is substantially faster. When it's time to make a change to their information, end users needn't worry about reconfiguring their client software, because the LDAP directory servers handle the data exchange for them.

Security and access control

LDAP provides for a complex level of access control instances, or ACIs. Because the access can be controlled on the server side, it's much more secure than security methods that work by securing data through client software.

With LDAP ACIs, you can do things like:

- Grant users the ability to change their home phone number and home address, while restricting them to read-only access for other data types (such as job title or manager's login).
- Grant anyone in the group "HR-admins" the ability to modify any user's information for the following fields: manager, job title, employee ID number, department name, and department number. There would be no write permission to other fields.
- Deny read access to anyone attempting to query LDAP for a user's password, while still allowing a user to change his or her own password.
- Grant managers read-only permission for the home phone numbers of their direct reports, while denying this privilege to anyone else.
- Grant anyone in the group "host-admins" to create, delete, and edit all aspects of host information stored in LDAP.
- Via a Web page, allow people in "foobar-sales" to selectively grant or deny themselves read access to subsets of the customer contact database. This would, in turn, allow these individuals to download the customer contact information to their local laptops or to a PDA. (This will be most useful if your sales force automation tool is LDAP-aware.)
- Via a Web page, allow any group owner to add or remove any entries from groups they own. For example, this would allow sales managers to grant or remove access for salespeople to modify Web pages. This would allow owners of mail aliases to add and remove users without having to contact IT. Mailing lists designated as "public" could allow users to add or remove themselves (but only themselves) to or from those mail aliases. Restrictions can also be based on IP address or hostname. For example, fields can be made readable only if the user's IP address begins with 192.168.200.*, or if the user's reverse DNS hostname maps to *.foobar.com.

This will give you an idea what's possible using access control with LDAP directories, but be aware that a correct implementation requires much more information than is given here. I'll discuss access control in greater detail in a future article.

That's it for now. I hope you've found this article useful. If you have comments or questions, send email to donnelly@ldapman.org.

28 april 2000

MENU
Home
Term of the Day
New Terms
Pronunciation
New Links
Quick Reference
Did You Know?
Categories
Tech Support
Technology Jobs
About Us
Link to Us
Advertising

Compare Prices: go
HardwareCentral

Talk To Us...
Submit a URL
Suggest a Term
Report an Error

internet.com
Developer
International
Internet Lists
Internet News
Internet Resources
IT
Linux/Open Source
Personal Technology
Small Business
Windows Technology
xSP Resources
Search internet.com
Advertise
Corporate Info
Newsletters
Tech Jobs
E-mail Offers

internet commerce
Be a Commerce Partner

LDAP

Last modified: Thursday, March 04, 2004

Short for *Lightweight Directory Access Protocol,* a set of protocols for accessing information directories. LDAP is based on the standards contained within the X.500 standard, but is significantly simpler. And unlike X.500, LDAP supports TCP/IP, which is necessary for any type of Internet access. Because it's a simpler version of X.500, LDAP is sometimes called *X.500-lite.*

Although not yet widely implemented, LDAP should eventually make it possible for almost any application running on virtually any computer platform to obtain directory information, such as email addresses and public keys. Because LDAP is an open protocol, applications need not worry about the type of server hosting the directory.

•E-mail this definition to a colleague•

*For internet.com pages about **LDAP** **CLICK HERE.** Also check out the following links!*

LINKS

=!= = Great Page!

LDAP 101: Glue Your Network's Pieces Together =!=
'The protocol nobody ever heard of' is gluing networks together with fast, flexible directory services and single sign-on for hassled users.

An introduction to directories and X.500
Contains an overview of directories with specific emphasis on the X.500 directory architecture. Also provides information on the LDAP standard and links to related information about directories, the X.500 standard, and directory services.

Related Categories
Directory Services
Internet
Internet Protocols

Related Terms
Active Directory
DEN
directory service
NDS
public-key encryption
TCP/IP

(Webopedia)
Give Us Your Feedback

An LDAP Roadmap and FAQ

Presents an annotated tutorial roadmap of LDAP documents and resources, including information about the IETF's directory service efforts, links to implementations, and links to existing LDAP/X.500-based directories.

LDAP resources

Contains links to LDAP news items, overview documents, client, gateway and server software, mailing list information, and related Web resources.

RFC 1777

Request for Comments document for Lightweight Directory Access Protocol (LDAP).

Yahoo!'s LDAP page

Yahoo!'s directory of LDAP.

1.1. What's LDAP ?

LDAP stands for Lightweight Directory Access Protocol. As the name suggests, it is a lightweight client-server protocol for accessing directory services, specifically X.500-based directory services. LDAP runs over TCP/IP or other connection oriented transfer services. LDAP is defined in RFC2251 "The Lightweight Directory Access Protocol (v3).

A directory is similar to a database, but tends to contain more descriptive, attribute-based information. The information in a directory is generally read much more often than it is written. Directories are tuned to give quick-response to high-volume lookup or search operations. They may have the ability to replicate information widely in order to increase availability and reliability, while reducing response time. When directory information is replicated, temporary inconsistencies between the replicas may be OK, as long as they get in sync eventually.

There are many different ways to provide a directory service. Different methods allow different kinds of information to be stored in the directory, place different requirements on how that information can be referenced, queried and updated, how it is protected from unauthorized access, etc. Some directory services are local, providing service to a restricted context (e.g., the finger service on a single machine). Other services are global, providing service to a much broader context.

1.2. How does LDAP work ?

LDAP directory service is based on a client-server model. One or more LDAP servers contain the data making up the LDAP directory tree or LDAP backend database. An LDAP client connects to an LDAP server and asks it a question. The server responds with the answer, or with a pointer to where the client can get more information (typically, another LDAP server). No matter what LDAP server a client connects to, it sees the same view of the directory; a name presented to one LDAP server references the same entry it would at another LDAP server. This is an important feature of a global directory service, like LDAP.

1.3. LDAP backends, objects and attributes

The LDAP server daemon is called *Slapd*. *Slapd* supports a variety of different **database backends** which you can use.

They include the **primary choice BDB**, a high-performance transactional database backend; LDBM, a lightweight DBM based backend; SHELL, a backend interface to arbitrary shell scripts and PASSWD, a simple backend interface to the passwd(5) file.

BDB utilizes Sleepycat Berkeley DB 4. LDBM utilizes either Berkeley DB or GDBM.

BDB transactional backend is suited for multi-user read/write database access, with any mix of read and write operations. BDB is used in applications that require:

- Transactions, including making multiple changes to the database atomically and rolling back uncommitted changes when necessary.
- Ability to recover from systems crashes and hardware failures without losing any committed transactions.

In this document I assume that you choose the **BDB database.**

To import and export directory information between LDAP-based directory servers, or to describe a set of changes which are to be applied to a directory, the file format known as LDIF, for LDAP Data Interchange Format, is typically used. A LDIF file stores information in object-oriented hierarchies of entries. The LDAP software package you're going to get comes with an utility to convert LDIF files to the BDB format

A common LDIF file looks like this:

```
dn: o=TUDelft, c=NL
o: TUDelft
objectclass: organization
dn: cn=Luiz Malere, o=TUDelft, c=NL
cn: Luiz Malere
sn: Malere
mail: malere@yahoo.com
objectclass: person
```

As you can see each entry is uniquely identified by a distinguished name, or DN. The DN consists of the name of the entry plus a path of names tracing the entry back to the top of the directory hierarchy (just like a tree).

In LDAP, an **object class** defines the collection of **attributes** that can be used to define an entry. The LDAP standard provides these basic types of object classes:

- Groups in the directory, including unordered lists of individual objects or groups of objects.
- Locations, such as the country name and description.
- Organizations in the directory.
- People in the directory.

An entry can belong to more than one object class. For example, the entry for a person is defined by the *person* object class, but may also be defined by attributes in the inetOrgPerson, groupOfNames, and organization objectclasses. The server's object class structure (it's schema) determines the total list of required and allowed attributes for a particular entry.

Directory data is represented as attribute-value pairs. Any specific piece of information is associated with a descriptive attribute.

For instance, the commonName, or cn, attribute is used to store a person's name . A person named Jonas Salk can be represented in the directory as

```
cn: Jonas Salk
```

Each person entered in the directory is defined by the collection of attributes in the *person* object class. Other attributes used to define this entry could include:

```
givenname: Jonas
surname: Salk
mail: jonass@airius.com
```

Required attributes include the attributes that must be present in entries using the object class. All entries require the objectClass attribute, which lists the object classes to which an entry belongs.

Allowed attributes include the attributes that may be present in entries using the object class. For example, in the person object class, the cn and sn attributes are required. The description, telephoneNumber, seeAlso, and userpassword attributes are allowed but are not required.

Each attribute has a corresponding syntax definition. The syntax definition describes the type of information provided by the attribute, for instance:

- bin binary.
- ces case exact string (case must match during comparisons).
- cis case ignore string (case is ignored during comparisons).
- tel telephone number string (like cis but blanks and dashes `-' are ignored during comparisons).
- dn distinguished name.

Note: Usually objectclass and attribute definitions reside on schema files, on the subdirectory *schema* under the OpenLDAP installation home.

Prev
How does LDAP work ?
Home
Up
Next
New versions of this document

3.8. Configuration File Example

The following is an example configuration file, interspersed with explanatory text. It defines two databases to handle different parts of the X.500 tree; both are BDB database instances. The line numbers shown are provided for reference only and are not included in the actual file. First, the global configuration section:

```
1.    # example config file - global configuration section
2.    include /usr/local/etc/schema/core.schema
3.    referral ldap://root.openldap.org
4.    access to * by * read
```

Line 1 is a comment. Line 2 includes another config file which contains core schema definitions. The referral directive on line 3 means that queries not local to one of the databases defined below will be referred to the LDAP server running on the standard port (389) at the host root.openldap.org.

Line 4 is a global access control. It applies to all entries (after any applicable database-specific access controls).

The next section of the configuration file defines a BDB backend that will handle queries for things in the "dc=example,dc=com" portion of the tree. The database is to be replicated to two slave slapds, one on truelies, the other on judgmentday. Indexes are to be maintained for several attributes, and the userPassword attribute is to be protected from unauthorized access.

```
5.    # BDB definition for the example.com
6.    database bdb
7.    suffix "dc=example,dc=com"
8.    directory /usr/local/var/openldap-data
9.    rootdn "cn=Manager,dc=example,dc=com"
10.   rootpw secret
11.   # replication directives
12.   replogfile /usr/local/var/openldap/slapd.replog
13.       replica uri=ldap://slave1.example.com:389
14.           binddn="cn=Replicator,dc=example,dc=com"
15.           bindmethod=simple credentials=secret
16.   replica uri=ldaps://slave2.example.com:636
17.           binddn="cn=Replicator,dc=example,dc=com"
18.           bindmethod=simple credentials=secret
19.   # indexed attribute definitions
20.   index uid pres,eq
21.   index cn,sn,uid pres,eq,sub
22.   index objectClass eq
23.   # database access control definitions
24.   access to attr=userPassword
25.           by self write
26.           by anonymous auth
27.           by dn.base="cn=Admin,dc=example,dc=com" write
28.           by * none
29.   access to *
```

```
30.             by self write
31.             by dn.base="cn=Admin,dc=example,dc=com" write
32.             by * read
```

Line 5 is a comment. The start of the database definition is marked by the database keyword on line 6. Line 7 specifies the DN suffix for queries to pass to this database. Line 8 specifies the directory in which the database files will live.

Lines 9 and 10 identify the database "super user" entry and associated password. This entry is not subject to access control or size or time limit restrictions. Please remeber to encrypt the rootpw using slappasswd.

Example: rootpw {SSHA}Jq4xhhkGa7weT/0xKmaecT4HEXsdqiYA

Lines 11 through 18 are for replication. See the Replication link for more information on these directives.

Lines 20 through 22 indicate the indexes to maintain for various attributes.

Lines 24 through 32 specify access control for entries in the this database. As this is the first database, the controls also apply to entries not held in any database (such as the Root DSE). For all applicable entries, the userPassword attribute is writable by the entry itself and by the "admin" entry. It may be used for authentication/authorization purposes, but is otherwise not readable. All other attributes are writable by the entry and the "admin" entry, but may be read by all users (authenticated or not).

The next section of the example configuration file defines another BDB database. This one handles queries involving the dc=example,dc=net subtree but is managed by the same entity as the first database. Note that without line 39, the read access would be allowed due to the global access rule at line 4.

```
33.     # BDB definition for example.net
34.     database bdb
35.     suffix "dc=example,dc=net"
36.     directory /usr/local/var/openldap-data-net
37.     rootdn "cn=Manager,dc=example,dc=com"
38.     index objectClass eq
39.     access to * by users read
```

Introduction to LDAP (Lightweight Directory Access Protocol)

By Manning Publications Co.

Go to page: Prev 1 2 3 4 5 6 Next

1.4.2 A new standard is born

In 1991, after a few false starts with other potential standards, LDAP was brought forth as a lightweight means for accessing the DAP-enabled directories of the X.500 world. The first set of standards, LDAPv2, were eventually defined and accepted by the Internet Engineering Task Force (IETF), an important standards body responsible for many important Internet standards, as RFCs 1777 and 1778.

These standards provided basic authentication, search, and compare operations, as well as additional operations for changing the directory. From the start, LDAP made X.500 more accessible, as intended. Figure 1.8 shows an X.500 server being accessed by an LDAP gateway service that is forwarding requests from an LDAP client.

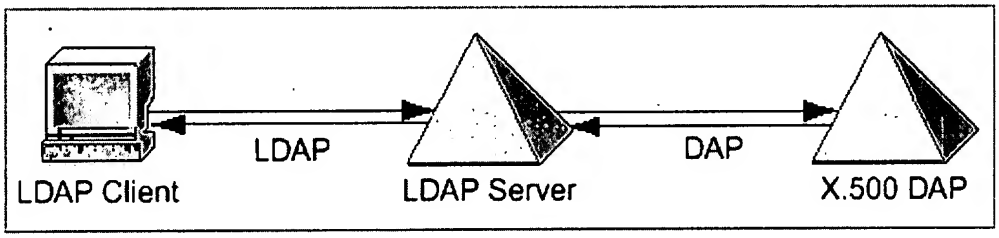


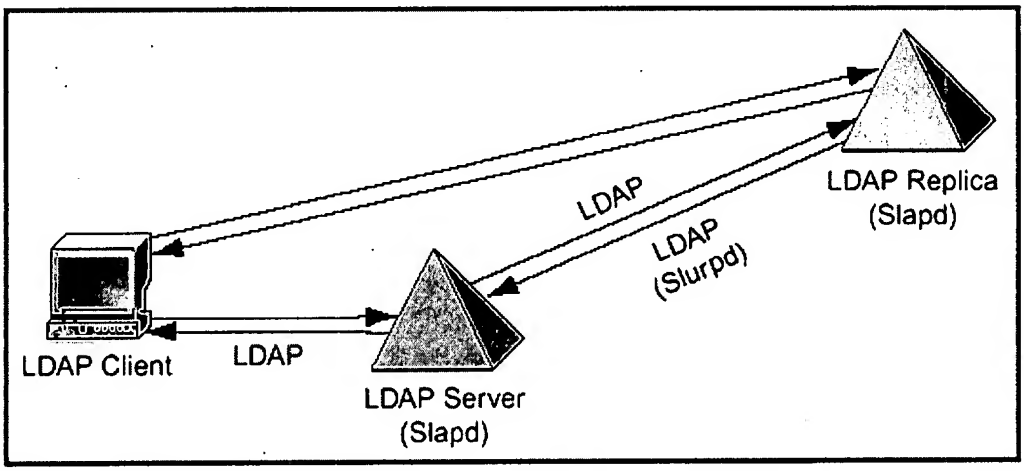
Figure 1.8
The X.500 client goes away, replaced by an LDAP client talking to an LDAP server. Here, the LDAP server acts as a gateway between LDAP-aware clients and DAP-aware X.500 DSAs.

Almost as important as the protocol itself was the release of a standard API and the production of a client development kit. For the first time, it was possible to access these servers programmatically without wandering knee-deep into an arcane protocol.

1.4.3 LDAP goes solo

As time went by, some people began to wonder what made X.500 so special in the first place. The University of Michigan, which had developed the reference implementation of LDAP, released a stand-alone server called Slapd that would allow the LDAP server to present data from a local data store rather than simply act as a gateway to an X.500 server.

Slapd was followed by a second service called Slurpd, which read the changes from one server and replicated those changes via the LDAP protocol to other Slapd servers. Figure 1.9 shows a typical stand-alone LDAP environment.



Click here for a larger image.

Figure 1.9
An LDAP client talks to a Slapd server. X.500 is no longer involved.

At this point, Netscape hired most of the original developers from the University of Michigan Slapd server to develop the Netscape Directory Server. Netscape, which was riding high with an incredible share of the Internet browser market, decided that networks would require directories and that LDAP, not X.500, should be the standard. Nearly 40 other companies announced support at that time, bringing LDAP the focus and support it needed to become the de facto standard for directory services.

1.4.4 LDAPv3

LDAP may have gained acceptance as a stand-alone service, but it was far from complete. Due primarily to its reliance on X.500 servers to provide the server-to-server communications, access control, and other functionality, LDAP was still only a skeleton of a full directory service by the mid-1990s.

Many interested parties pushed forward with the development of the next generation of the LDAP standards. In December 1996, the new version was published as RFCs 2251 to 2256. These new specifications covered items including the protocol itself, mandatory and optional schema, and LDAP URLs. A set of standard authentication mechanisms and a standard for session encryption were added to the list of core specifications in 2000. Figure 1.10 shows the core specifications that make up the LDAP standard.

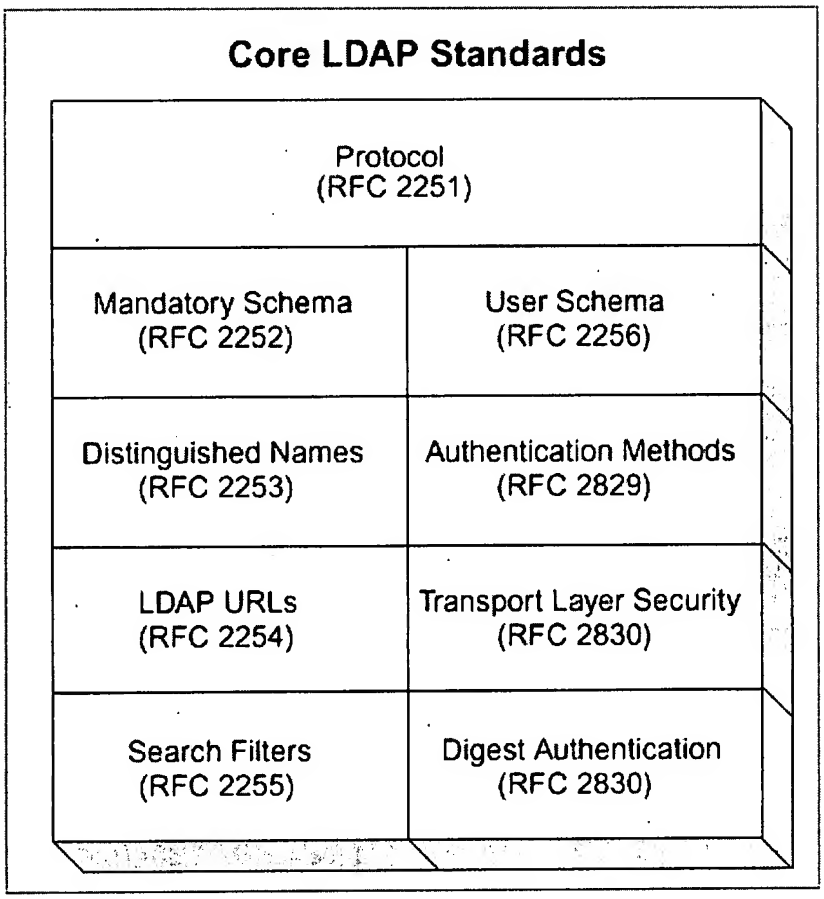


Figure 1.10
The IETF has been the primary standards body for most of the existing LDAPv3 specifications. This figure shows a list of published RFCs that are considered the core LDAP standards.

1.5 LDAP REVISIONS AND OTHER STANDARDS

LDAPv3 was considered a great leap forward in several key areas, but it takes more than a protocol to make a directory service successful. It is now up to several standards bodies and industry consortia to enhance the LDAP core specifications and build a framework that allows directories from different vendors to interoperate, or at least share some of the most crucial information in a standard way, and play a more pivotal role in e-business. Figure 1.11 shows some of the many standards bodies and industry consortia that shape directory standards and define best practices in deployment and management.

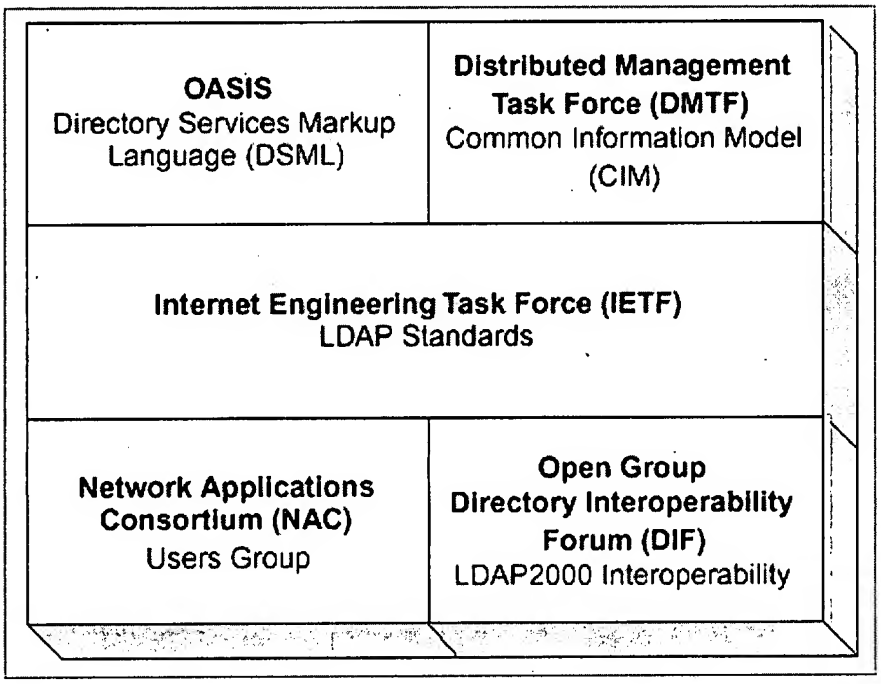


Figure 1.11
Many industry consortia and standards bodies are involved with LDAP and related standards, but most have a narrow focus.

1.5.1 Replication and access control

Version 3 of the LDAP protocol was greatly improved from version 2, but lacked two important items: replication and access control. The IETF has created workgroups to deliver these missing pieces and others, as shown in figure 1.12.

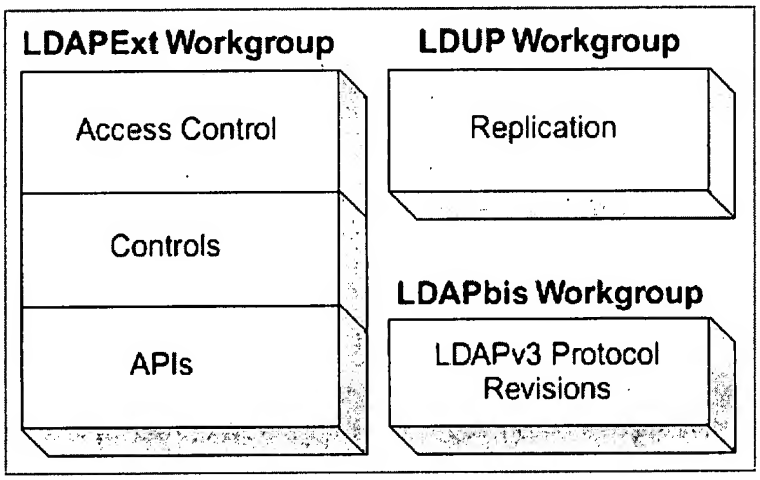
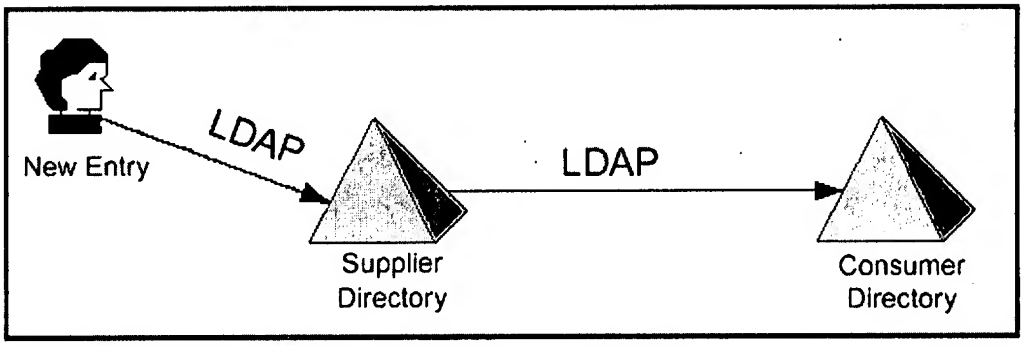


Figure 1.12
IETF workgroups are trying to fill in the gaps left after the initial publication of LDAPv3.

Lack of a standard replication process has since become an interoperability nightmare as each LDAP server vendor implemented its own proprietary solution. Many products use simple LDAP protocol operations to distribute data as shown in figure 1.13. However, even those solutions using the LDAP protocol sometimes require proprietary controls or attributes.

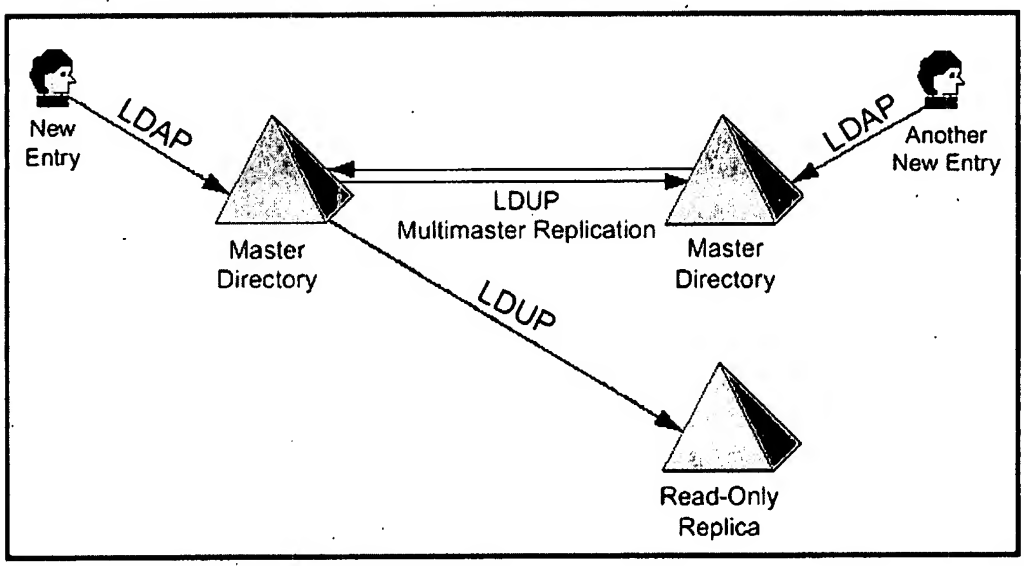


Click here for a larger image.

Figure 1.13
Supplier-to-consumer replication exists in some products using the LDAP protocol. Unfortunately, most need to use proprietary attributes or controls to get around current limitations in the specifications.

Many parties recognized that replication was critical to obtaining scalability, redundancy, and other important benefits. To resolve this issue, the Lightweight Directory Update Protocol (LDUP) working group was created within the IETF. At the time of this writing, the group has completed draft documents detailing requirements, a model for meeting those requirements, conflict resolution processes, and a protocol specification. The use of replication is discussed further in chapter 6.

In addition to the supplier-consumer model of replication available in most existing directory servers, LDUP was chartered with allowing for multiple directory masters for the same information, which is shown in figure 1.14. It also documents a process for resolving conflicts that may arise when different and potentially conflicting changes are made independently to the same entry on each master. In addition, LDUP defines a protocol that can be used for both supplier-initiated and consumer-initiated replication.



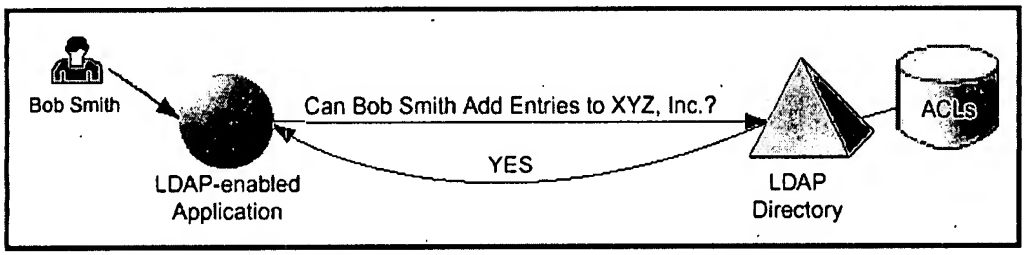
Click here for a larger image.

Figure 1.14
Multimaster replication will allow changes to the same directory tree in multiple

directories.

Security was further along in some respects. The Simple Authentication and Security Layer (SASL), originally developed for the Internet Mail Access Protocol (IMAP), was added as a core LDAP standard early on as a way to negotiate an appropriate type of client and/or server authentication and even session encryption.

Developing a standard for access control has proven to be much more time consuming and has produced fewer results. As shown in figure 1.15, such a standard will allow a server to determine if an authenticated entity should be able to read or update a particular entry or an entire portion of the directory.



Click here for a larger image.

Figure 1.15
LDAP access control standards will include a mechanism for determining in advance whether an operation will be permitted.

The task of creating such a standard fell into the hands of the LDAP extensions (LDAPEXT) workgroup within the IETF. This workgroup was formed to handle any extensions needed to the LDAPv3 standards outside of replication. As this book is being written, most activities of the LDAPEXT workgroup have been moved to individual submissions and will likely become an informational RFC rather than a full standard. Some aspects of access control may be pursued as part of the interoperability requirements for replication.

To understand why access control might be bundled with the replication workgroup, think about the fact that any replication of information outside a vendor's products will render that data insecure—other vendors will not know the access control rules of the source data. Any practical solution for replication is dependent on a standard for access control. We will look at access control further in chapter 13 when we discuss directory security in more detail.

Go to page: Prev 1 2 3 4 5 6 Next

Introduction to LDAP (Lightweight Directory Access Protocol)

By Manning Publications Co.

Go to page: Prev 1 2 3 4 5 6 Next

1.5.2 Directory Enabled Networking

As computer networks evolve to support more variety and depth of services, the complexity of network management increases accordingly. Most network devices, including routers and switches, have traditionally been configured using command-line shells. Although this configuration enables relatively consistent management of a single device, it does nothing to simplify the coordination of configurations across large numbers of devices. Such coordination is critical when you're enabling guaranteed quality of service and other offerings that span multiple devices.

Directory Enabled Networking (DEN) provides a way for devices to configure themselves based on information in a directory. Originally an initiative from Microsoft and Cisco, DEN is now part of the CIM defined by the DMTF.

CIM is a set of object-oriented, implementation-neutral schemas that represents logical and physical attributes of hardware and software. The DMTF, rather than being protocol architects like the IETF, focused primarily on creating common object definitions that allow two CIM-aware applications to store and use information consistently.

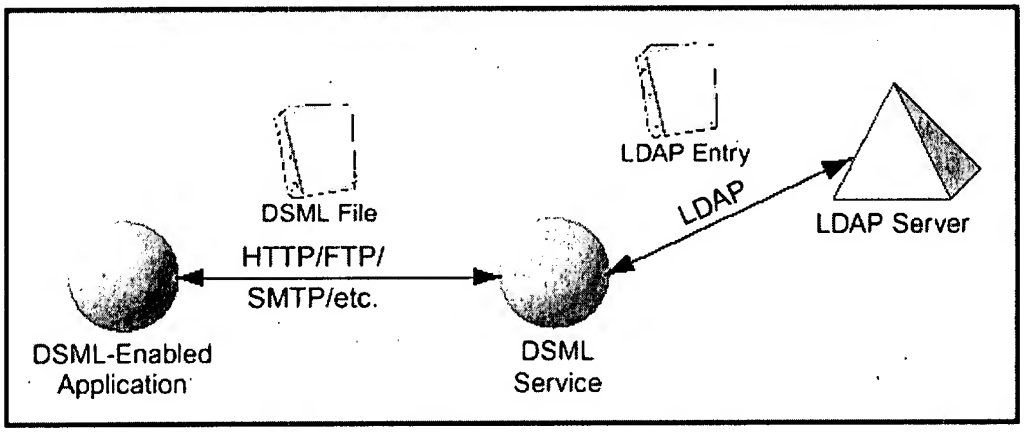
Contrary to popular belief, CIM and DEN are not LDAP-specific information models, but are instead "meta" models that can be specialized for a number of environments, of which LDAP is one. XML is an example of another way that CIM objects can be represented.

Momentum behind DEN as the killer application that would drive directories has died down to an extent over the l ast few years, and most of the work around directories has moved to identity management solutions. In this book, we will not focus on DEN as a specific application due to the current lack of software and hardware that can truly exploit this technology.

1.5.3 XML and directories

The eXtensible Markup Language (XML) is an industry standard language used to define structured documents. It offers a set of common tags for defining data about data, or *metadata*. This metadata can be used to describe particular document types. Instances of documents implementing these types can then be shared and used by XML-aware applications.

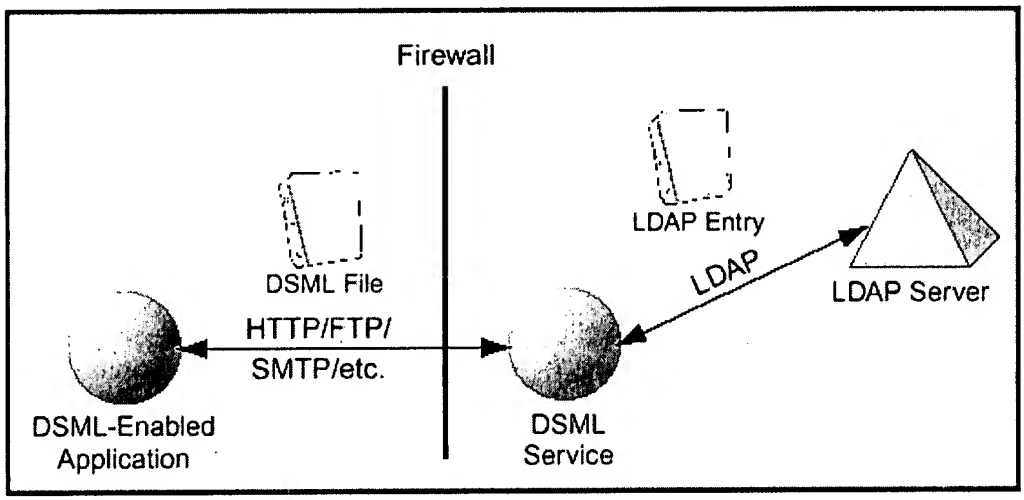
DSML is an XML document type that can be used to create structured documents representing information in a directory service. This information represented in DSML can include both directory entries and schema information. DSMLv2 extends the specification to cover the representation of directory operations. Documents conforming to these standards can be exchanged using non-directory protocols like HTTP, as shown in figure 1.16. Many new services that support DSML are becoming available from both large vendors (Sun and Microsoft) and startups.



Click here for a larger image.

Figure 1.16
Here a DSML-enabled application talks to a DSML service that acts as an intermediary between an LDAP server and the DSML-enabled application.

DSML is most useful in applications that are already XML enabled. These include most modern application servers. DSML is especially useful in cases where direct access to the directory would normally not be permitted. For example, consider a situation in which a firewall is blocking all traffic except HTTP. To get around this limitation, a DSML encoding of a directory entry can be transmitted over the HTTP protocol for interpretation and presentation. Such a situation is shown in figure 1.17. Emerging standards like Simple Object Access Protocol (SOAP) make it clear that LDAP will not be the only standard for sharing directory information in the future.



Click here for a larger image.

Figure 1.17
DSML is useful for sharing directory information across firewalls that might limit direct access to directories.

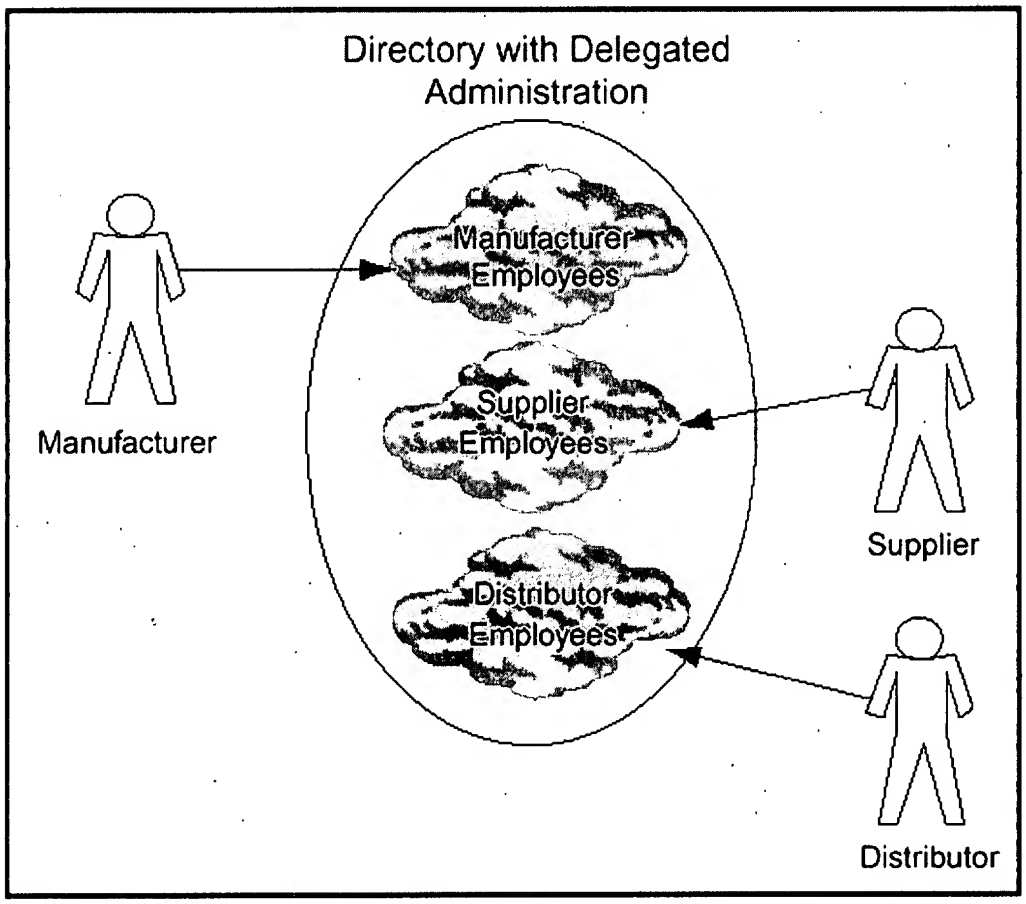
1.6 DIRECTORY MANAGEMENT

Despite the importance of having well-defined standards, it is rarely the reason for a directory services—related project to fail. Rather, the biggest headache with most new directory deployments is proper management of information in the directory. In the days when enterprise

directories were used primarily for storing white pages information, it was often adequate to simply import information into the directory periodically from other, more authoritative data sources. Due to the lack of sophisticated management tools, there wasn't much choice.

Today, directory management tools for users and groups are much more sophisticated. In addition to giving a central administrator the ability to change information about objects in a directory, these tools typically allow for delegation of administrative duties and even user self-management, where appropriate.

This ability to distribute administration works well in intranet and Internet environments, but it is especially critical in *extranet* environments where multiple organizations are working together, potentially using the same applications and data. In such environments, the segmentation of administration and access is very important (see figure 1.18).



Click here for a larger image.

Figure 1.18
Directories can be segmented such that administration can be delegated to business partners. Such separation may be logical rather than physical.

For example, a car manufacturer with just-in-time manufacturing facilities needs to give its business partners access to certain systems in its extranet. Access to applications on the extranet is controlled based on identities in each of its distributors and component suppliers. Tracking by identity offers audit trails, which will deter a random individual from anonymously ordering unauthorized parts.

The problem is, in addition to the employees at the company, such an extranet environment including suppliers and distributors may include hundreds of thousands, if not millions, of users. Trying to manage all these users centrally would be an incredible effort.

By segmenting users by company and other means, you can push administration of identities to primary contacts within each of the business partners, thereby reducing administrative overhead. Aside from reducing administration costs, this approach also ensures better accuracy by pushing identity management closer to the identities being managed.

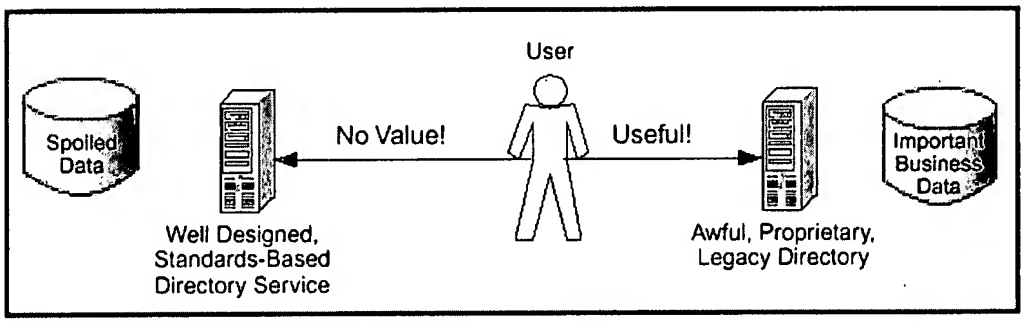
Information that is not related to identities and groups can still be difficult to manage with off-the-shelf products. This is the case primarily because little attention has been paid to other advanced uses of directories, such as DEN, which require management of more exotic information.

In chapter 7, we will look at managing all types of directory entries, complete with example applications to reduce manual data entry and allow some degree of user self-management.

1.7 DIRECTORY INTEGRATION

Many organizations spend months designing the schema, entry naming, and other related aspects of an enterprise directory service without considering the need for integration with existing information repositories. What usually results is a well-designed, standards-based directory service that contains stale information and is nearly useless.

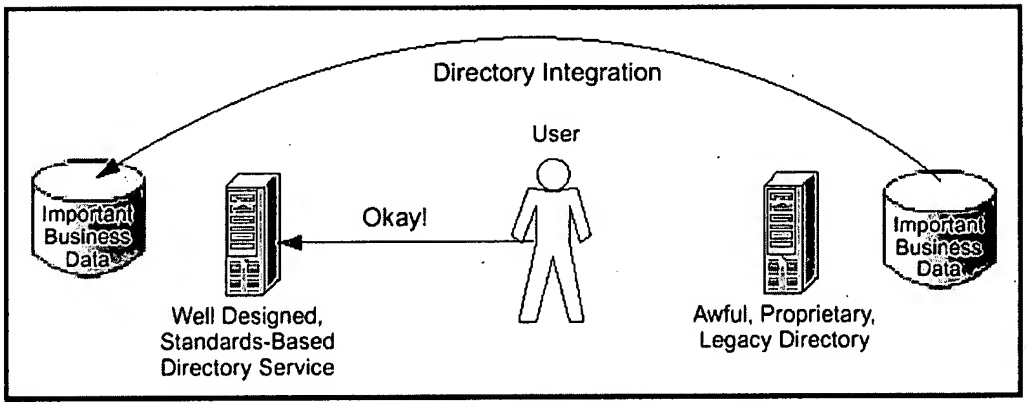
Meanwhile, legacy data stores that contain mission-critical information continue to thrive because they contain fresh information, although in a way that is often inconvenient to access from new applications and nearly impossible to access from off-the-shelf applications without substantial custom development. Figure 1.19 shows how this typical scenario plays out.



Click here for a larger image.

Figure 1.19
Data in legacy systems is nearly always more useful than data in poorly integrated new systems.

By designing and implementing an appropriate level of directory integration between legacy data stores and the new directory service, you can dramatically increase the value of the new directory (see figure 1.20).

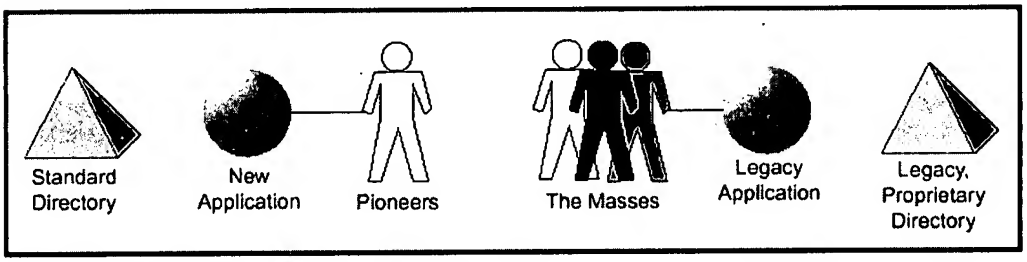


Click here for a larger image.

Figure 1.20
Some level of directory integration is important in increasing the value of applications using new directory services.

Directory integration is far more complicated than simply synchronizing everything from a legacy data store into a newly created directory. It demands that you evaluate the needs of applications that depend on both new and legacy data stores. In many cases, both new and legacy applications that utilize the respective data stores. Very often, these applications need access to some set of the same information.

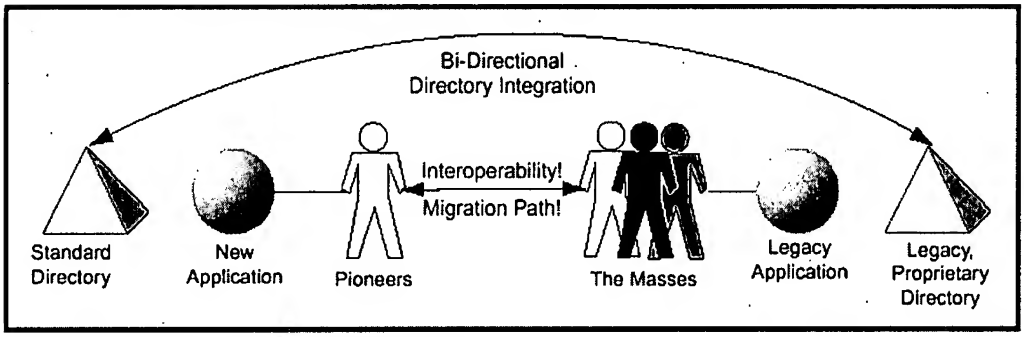
Without any directory integration, it is often difficult to get more than a small group of pioneers to quickly adopt the new applications. A new application may have substantially better functionality, but without the proper data it will be difficult to move the masses that use the legacy applications to the new environment. This issue is demonstrated in figure 1.21.



Click here for a larger image.

Figure 1.21
It is difficult to move the masses to new applications based around a standards-based directory when important information still resides only in a legacy directory.

By using integration techniques, such as synchronization, you can create a high degree of interoperability between the two environments. This approach, shown in figure 1.22, provides the necessary data flow between the two directories, offering a relatively easy migration path to the new environment. It also ensures that the information in both environments is consistent.



Click here for a larger image.

Figure 1.22
Synchronization is often necessary to offer a migration path from legacy to new applications or interoperability where legacy applications will not be migrated.

Consolidating these two environments can vastly simplify management. For example, you may find a way for a Unix-based system to use the same directory as your white pages application to store password information.

Go to page: Prev 1 2 3 4 5 6 Next

Introduction to LDAP (Lightweight Directory Access Protocol)

By Manning Publications Co.

Go to page: Prev 1 2 3 4 5 6

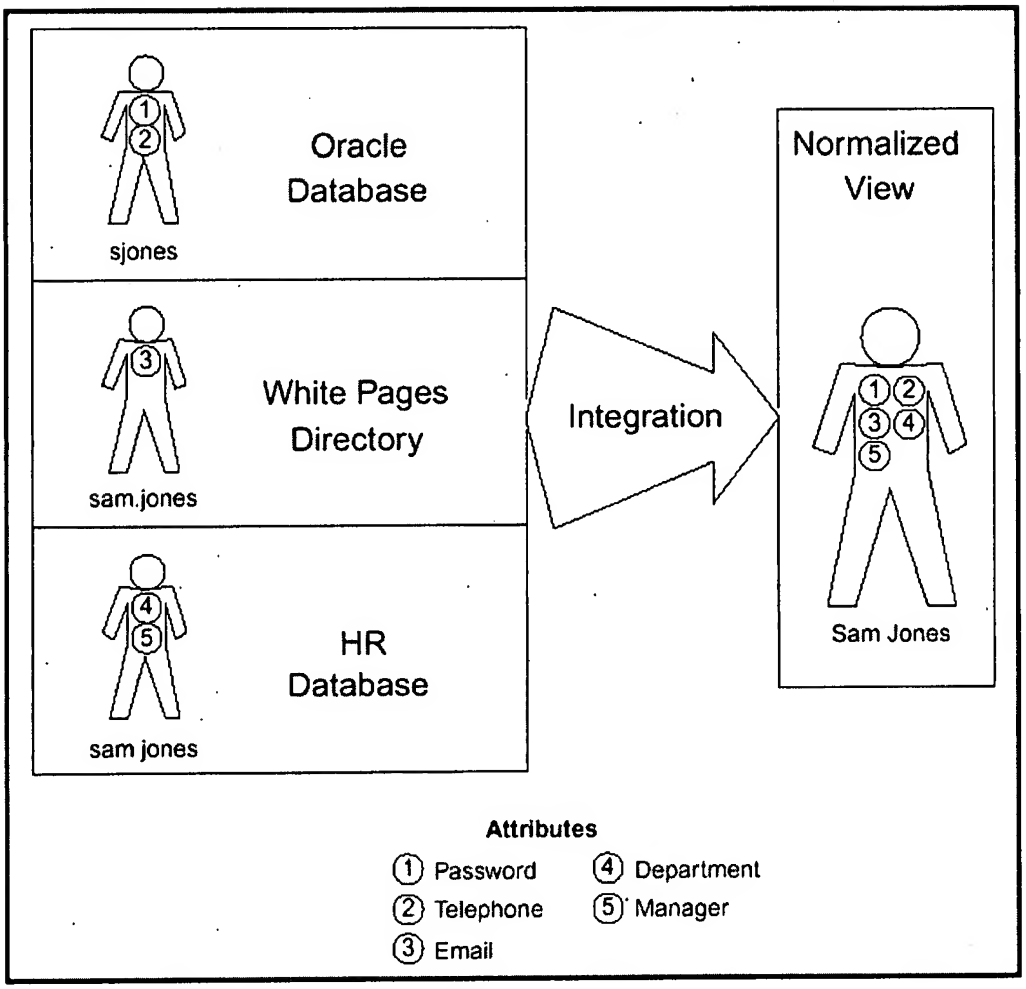
However, not every connected data store is a candidate for consolidation. Take, for example, a human resources application that relies on a set of database tables to store information. It may not make sense from an application functionality perspective for that particular application's data store to be consolidated into an enterprise directory. Some of the information may fit better in relational databases for the reasons we stated in section 1.2.1, whereas other information may not be a good candidate for synchronization because of privacy concerns. So, instead of attempting to directly replicate everything from human resources into the directory, you need a form of intelligent synchronization.

In the area of identity management, directory integration almost always seems like a great idea in theory. For example, the management of users' computer accounts in a particular organization from hire to fire demonstrates the value of synchronization and other advanced integration technology.

Today, it is often necessary to touch multiple data repositories to commit a single change uniformly to all the places that store information about a person. These changes are usually performed by different application and system administrators. In more mature environments, changes may be synchronized with scripts to facilitate this process. When administrators do not coordinate their changes, or if an automated synchronization script fails, the data repositories are no longer synchronized, and at least one of the repositories will contain stale data.

If this stale data is simply a telephone number, the impact is probably minimal. However, if an account must be deleted or suspended due to an employee's termination, the data repository with stale data is at risk from the terminated employee. If the stale data resides in an enterprise directory that is used for authenticating and authorizing users to all non-legacy systems and applications, this one failed change can potentially put the organization's entire intranet at risk. Proper directory integration is key to reducing these types of risks. For this reason, it is important to spend an adequate amount of time planning for integration.

A general integration planning process entails identifying which data elements exist in each existing data source, selecting those that should be shared, and mapping between the source and destination schema (see figure 1.23).



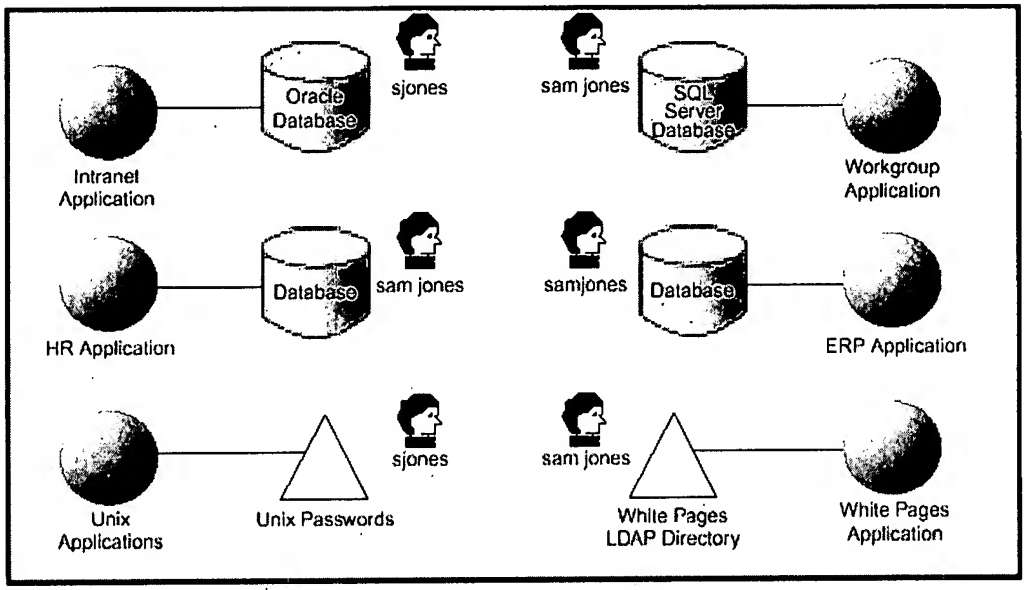
Click here for a larger image.

Figure 1.23
Multiple data repositories typically store information about a person. Deciding which attributes come from where and mapping them to a normalized schema is an important part of any directory integration process. Note that the word normalized here should not be confused with database normalization rules.

This process and ways of implementing it are described in detail in chapter 7.

1.7.1 Integration via metadirectories

We cannot emphasize enough that the consolidation of all data repositories into a single enterprise directory within even the smallest of organizations is not likely to happen in our lifetimes. Even if it were possible to rewrite every legacy application to use a single standard, different directory and database software is better for different tasks. As shown in figure 1.24, this leads to many different environments within an organization that have different variations of the same user.



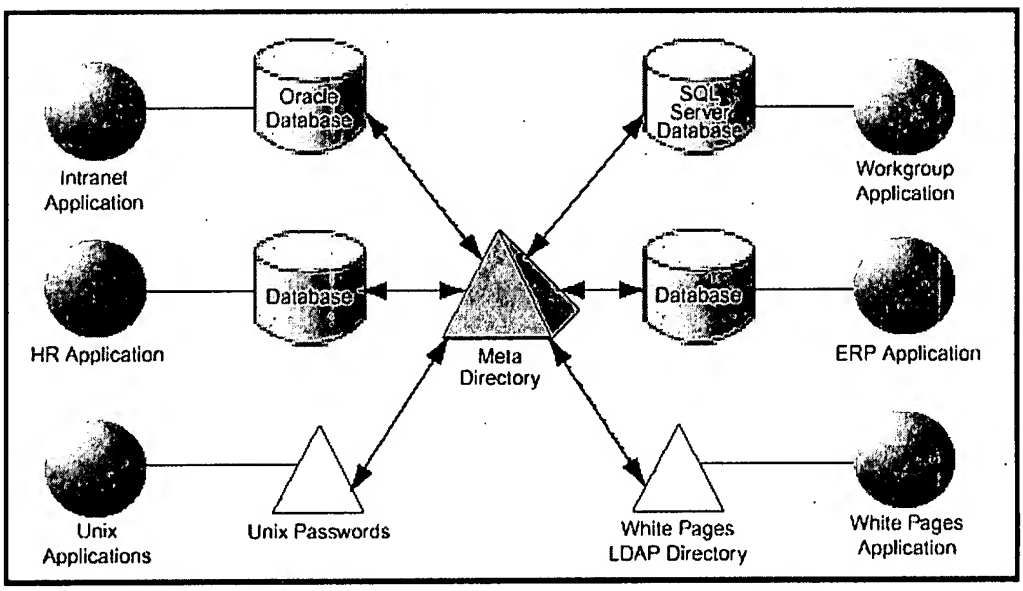
Click here for a larger image.

Figure 1.24
Different applications have different data repository requirements. It is not likely that a single data store could accommodate all of them.

In the past few years, a new breed of applications called *metadirectories* has come to market to remove some of the burden associated with directory integration. Although it may sound like yet another directory, a metadirectory is really a sophisticated directory integration toolkit.

You can use metadirectories to connect and join information between data sources, including directories, databases, and files. The connection process usually involves identifying changes in each data source. Such a connection may be real-time monitoring of changes using a direct access method into the connected data store, an occasional scan of a file-based list of changes, or a review of a full export from the connected data store.

The join process is much more complicated and usually involves several steps. Its most important job is determining that an object in one data source is the same as an object in a second data source. This aggregation of information from multiple data sources is one of the most important features of a metadirectory and the heart of the join process. Other tasks performed by a metadirectory may include unification or mapping of schema and object names, filtering unwanted information, and custom processing and transformation of data. Figure 1.25 shows a relatively logical view of how a metadirectory might work to provide a linkage between key enterprise information repositories.



Click here for a larger image.

Figure 1.25
Metadirectories provide advanced integration capabilities between different types of data stores.

With careful planning, you can create an environment in which users can be created at a single point. Then, the metadirectory service will instantiate a subset of the users' information in other connected data stores automatically, or with very little manual intervention. The actual point of instantiation may be managed by another type of software that handles the workflow needed by this process. Such software is called *provisioning software*.

For example, if PeopleSoft, white pages, and an Oracle database all use a telephone number, you would like that telephone number to be entered once and propagated to the other data stores. Metadirectories must also handle environments where both Oracle and PeopleSoft would be able to master new changes depending on business rules.

Metadirectories are also proving to be popular in extranet environments where two or more organizations have their own directories and want to share a portion of them with business partners or vendors. Figure 1.26 shows an extranet environment where the addition of Joe Distributor might be propagated to the manufacturer using metadirectory technology.

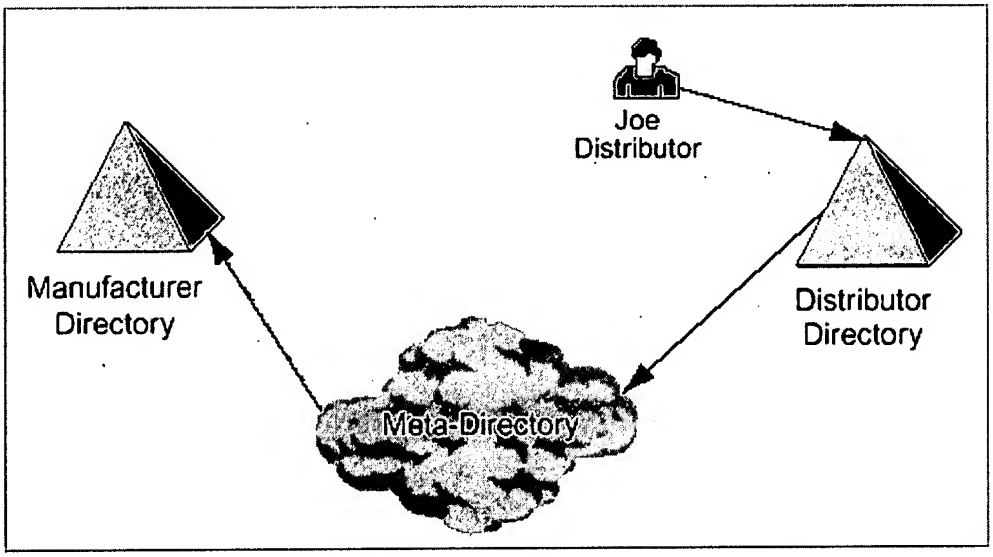
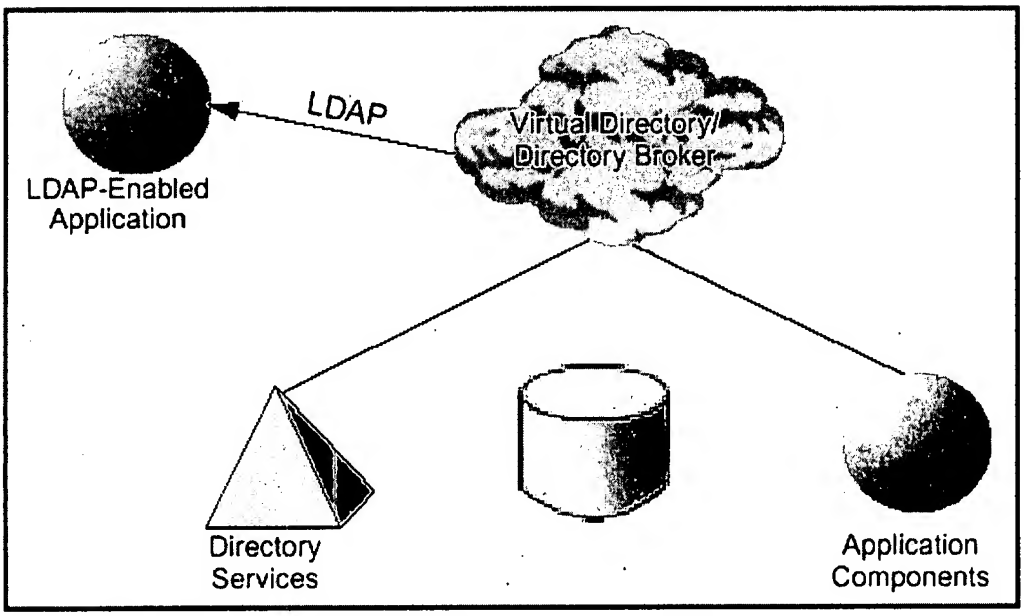


Figure 1.26
A user is entered into the distributor directory. The metadirectory detects a change and propagates it to an appropriate location within the manufacturer directory.

It is beyond the scope of this book to offer an in-depth look at metadirectory products. However, directory integration is critical, and some of the functionality provided by metadirectory products can be performed with a general scripting language. We discuss such techniques in detail in chapter 6.

1.8 INTEGRATION AND FEDERATION VIA VIRTUAL DIRECTORY TECHNOLOGY

Usually, metadirectories involve the creation of a new, physical directory, the contents of which are based on an aggregation of multiple information sources. One emerging alternative to metadirectory technology is *virtual directory* technology, sometimes called *directory federation* technology. This technology attempts to provide real-time directory access to other types of data stores, such as relational databases and memory-based application components. To visualize this process a bit more easily, think of the virtual directory as a kind of proxy server: the application speaks LDAP to the virtual directory software, and the virtual directory software grabs the data directly from the legacy data store by speaking its native tongue. Figure 1.27 shows a directory-enabled application accessing a virtual directory service that is providing data from existing directories, databases, and application components.



Click here for a larger image.

Figure 1.27
Virtual directories (sometimes called directory federators) accept directory requests and transform them into requests for potentially non-directory information.

Virtual directory technology is not as easy as it may sound. Each underlying data store has its own query language and information model. The virtual directory must find ways to optimize queries and map between LDAP and non-directory information models.

At this time, virtual directory technology is in its infancy, as metadirectories were a few years ago. However, it is emerging as another useful tool for providing a unified view of information to LDAP-enabled applications. It is the only way to view information in many kinds of existing repositories using directory protocols in real time.

1.9 WHY THIS BOOK?

People who have worked with directories know that installing and configuring most directory server software is generally the easiest part of a directory deployment. Writing simple applications to query the directory and use the results is also quite easy, once you understand the basics. Trouble begins to brew when it becomes necessary to keep the information in the directory up to date through both front-end data management and back-end integration with other data sources. This book focuses on making your directory deployments more successful through advanced application and interdirectory integration.

Consider that every element of data stored in a directory must be placed into the directory at some point. You can leverage the data that already exists in other repositories, someone can enter it into the directory through an administrative interface, or the data can be generated by an application. In many environments, all these tasks may need to happen to create a suitable directory service. Figure 1.28 shows some of these different techniques for moving information into the directory.

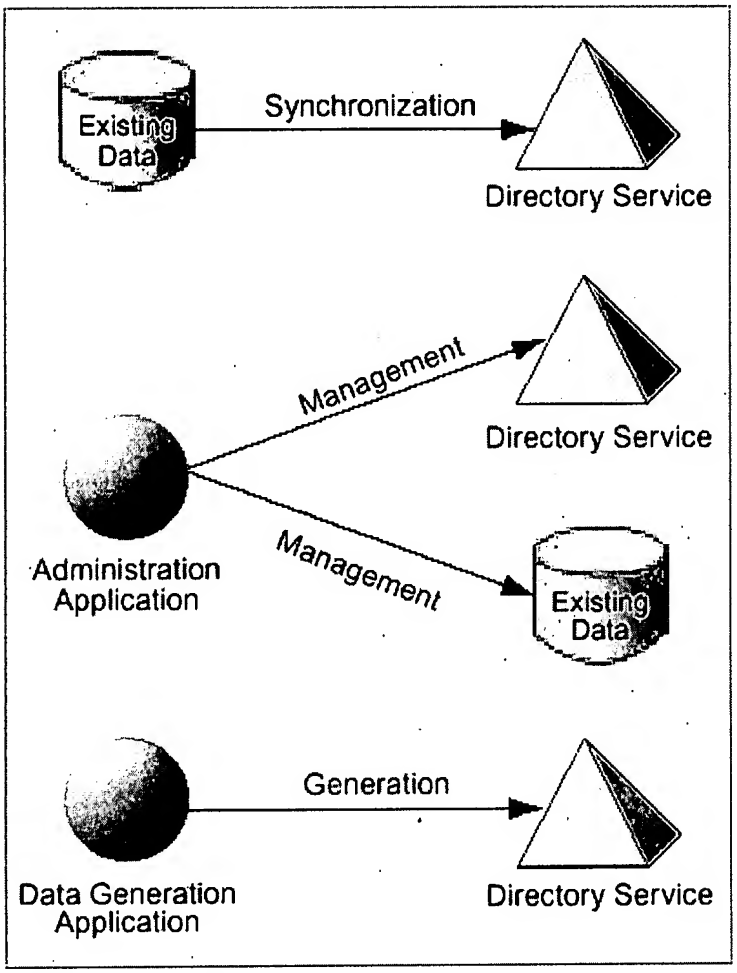


Figure 1.28
Data in directories is synchronized with existing data stores, managed through administration applications, and/or generated in some way.

For new and experienced directory service managers charged with deploying or managing a directory service, these management and integration issues are clearly the biggest challenge. Not having the right information, or having stale versions data, dilutes the value of the directory to all applications that leverage it.

Directory management involves having the right tools and tying in the right information from other, often authoritative, sources of data. In this book, we'll focus on practical solutions to common directory management problems. We will look at Perl code for administration interfaces, directory synchronization, and directory migration. The entire second part of the book is devoted to this topic.

Directory-enabled applications let you use all the information you've been collecting in directories. After all, why collect data if it nobody wants to use it? We'll look at ways to leverage LDAP in a variety of application environments with source code in Java. You'll find that such application integration is key to having a useful and important directory that people want to keep current.

With the information in this book, you'll have information flowing through your directories with much less perspiration. Servers that support the LDAP standard can provide a wide variety of functionality to a properly enabled application. This book aims to help you manage your LDAP directories and enable your applications, both new and existing, to support these directories.

1.10 SUMMARY

The LDAP standard for accessing directory services is important to software developers and system administrators. It can be used through LDAP-enabled applications and various APIs.

A number of different directory services have come into existence in the past few decades; LDAP was derived from another popular standard called X.500. These directory services provide everything from white pages to application security.

Management and application integration are the two biggest issues people tend to encounter when deploying directory services. You can address these issues many ways, as the second and third parts of this book explain.

The IETF has been the driving force behind the core LDAP specifications and many enhancements. Its most important current work is related to replication and access control. Other industry consortia and standards bodies are important in developing LDAP server and application interoperability guidelines, as well as standards that represent data from the LDAP information model in XML.

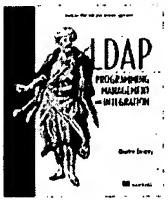
Metadirectories provide synchronized integration between multiple data repositories, and virtual directories provide real-time integration between applications and existing data via directory protocols. Provisioning tools allow for manual management of the information in directories. Each of these types of tools plays an important role in a well-rounded directory service.

In the remainder of part 1, we will focus on the LDAP standards in more detail, and discuss how to use LDAP tools to communicate with a directory server.

About the Author

Clayton Donley, the co-author of a number of open-source LDAP modules for Perl and Apache, is an independent consultant based in the Chicago area. His clients include Netscape, GTE, and ABN-AMRO. Prior to going independent, he spent seven years in various information technology roles working for Motorola in both the Chicago area and the Asia-Pacific region.

Source of this material



This is **Chapter 1: Introduction to LDAP** from the book ***LDAP Programming, Management and Integration*** (ISBN:1-93011-040-5) written by Clayton Donley, published by Manning Publications Co..

To access the full Table of Contents for the book.

Other Chapters from Manning Publications:

Struts in Action: Developing Applications with Tiles